EFTR-HGNet: An Efficient Rescheduling Method of Edge Service Tasks in Fault Scene

Lu Yao, Yan Wang, *Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—In edge computing environments, service reliability is often threatened by the sudden failure of edge nodes due to harsh deployment conditions, leading to task interruption and performance degradation. To address this challenge, EFTR-HGNet is proposed as a novel task rescheduling framework tailored for edge-fault scenarios. It leverages heterogeneous graph neural networks with a Transformer-based architecture to achieve cost-efficient task migration and adaptive decision making. Specifically, the rescheduling problem is formulated as a Markov decision process (MDP), and a 3-D fault-aware state representation that jointly encodes task attributes, resource availability, and dynamic failure status is introduced. To model the complex relationships between failed tasks and heterogeneous edge resources, a heterogeneous Transformer (HG-Trans) network is designed, which performs two-stage embedding over the constructed graph, enabling context-aware rescheduling decisions to be made by the agent. By optimizing both the policy and value functions within an Actor-Critic reinforcement learning framework, our method achieves a favorable balance between minimizing the overall Makespan and maximizing the task rescheduling success rate. Evaluated against strong baselines like heterogeneous earliest completion time first algorithm (HEFT), task replication and cluster-based scheduling algorithm (TDCA), and FixDoc, EFTR-HGNet demonstrated superior performance, achieving a Makespan reduction of at least 11.11% and a 4.20% increase in task rescheduling success. These results highlight its robustness and practical potential for fault-prone edge computing systems.**

***Index Terms*—Edge computing, fault tolerance, heterogeneous graph, task rescheduling.**

I. Introduction

With the rapid development of mobile edge computing (MEC) technologies, computing tasks are moving from centralized cloud processing to edge devices close to the data source [1] to reduce response times, data transmission latency, and improve computing efficiency. However, edge computing environments face many challenges due to their inherently distributed and heterogeneous nature, especially in terms of device failure and uncertainty [2]. Compared with traditional centralized computing architecture, edge nodes are often resource-constrained devices. These devices are susceptible to problems, such as hardware damage, power consumption limitations, and unstable network connections [3] which will have an impact on the task being performed, and even affect the stability of the entire system. Therefore, how to effectively recover the interrupted task and minimize the recovery cost becomes the key issue to ensure the performance of MEC system in the fault scenario.

The previous scheduling methods mainly include static scheduling and dynamic scheduling [4]. The static scheduling method determines the scheduling scheme in advance before the task starts to execute and does not consider the changes during the run time [5], so it is difficult to deal with dynamic faults. The dynamic scheduling method is adjusted according to the actual situation of the running time and has better adaptability [6]. However, how to migrate the interrupted task to other nodes with minimal cost while maintaining the task dependency after the failure in the MEC environment is still an unsolved challenge. When the edge device fails, not only the task execution success rate affected by the fault should be considered, but also the task recovery time after the fault should be minimized. In this case, it is necessary to develop a rescheduling method that can sense task dependencies and environment state when a fault occurs and migrate the failed tasks to other edge devices for execution at minimal cost.

We propose an edge-fault scenario task rescheduling method based on heterogeneous graph neural networks to solve the dynamic task scheduling in edge-fault scenarios. Our optimization goal is to maximize the efficiency and reliability of the edge system for task processing. Specifically, our strategy is designed to minimize Makespan and task response delays while maximizing the success rate of fail-interrupt task execution. The main contributions of this article are as follows.

1) *Heterogeneous Information State Representation:* A heterogeneous graph structure is adopted for state representation, effectively integrating multidimensional heterogeneous information, such as task dependencies, resource states, and fault information with low graph density. A comprehensive state representation model for capturing the complex nonlinear interactions among tasks, resources and faults is constructed. A fault-tolerant

Received 1 May 2025; revised 7 June 2025; accepted 21 July 2025. Date of publication 4 August 2025; date of current version 9 October 2025. This work was supported in part by the Natural Science Foundation of China under Grant 62162047; in part by the Key Research and Development and Technology Transfer Program of Inner Mongolia Autonomous Region under Grant 2025KJHZ0030 and Grant 2025YFHH0110; in part by the Natural Science Foundation of Inner Mongolia under Grant 2023MS06017 and Grant 2023ZD18; in part by the Inner Mongolia Youth Science and Technology Talents support project under Grant NJYT24034; in part by the Inner Mongolia Autonomous Region higher Education Carbon Peak and Carbon Neutral Research Project through Ministry of Education under Grant STZX202322; and in part by the Reform and Development of Local Universities (Disciplinary Construction). *(Corresponding author: Yan Wang.)*

Lu Yao and Yan Wang are with the College of Computer Science, Inner Mongolia University, Hohhot 010021, China (e-mail: 32209162@mail.imu.edu.cn; cswy@imu.edu.cn).

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/JIOT.2025.3595407

mechanism is introduced in the state representation to support the adaptive adjustment and recovery of interrupted tasks, improving the intelligent decision-making level and robustness of the scheduling system.

2) *Neural Network Architecture Based on (HG-Trans):* A neural network architecture based on HG-Trans is constructed. Information is extracted from the heterogeneous state graph through a two-stage embedding process (node embedding and operation embedding). The local and global feature relationships between heterogeneous nodes and edges are effectively handled to provide a comprehensive state-aware basis for dynamic online decision-making. Meanwhile, an optimization objective based on the balance of dual functions of strategy and value is designed, aiming to jointly minimize task completion time and maximize rescheduling success rate.
3) *Comprehensive Experimental Validation:* Through extensive experiments in the edge simulation environment, the effectiveness of EFTR-HGNet is demonstrated in comparison with other strong baseline scheduling algorithms, particularly in terms of makespan, task response delay, and task rescheduling success rate.

II. Related Work

Task scheduling aims to assign tasks to appropriate computing nodes based on task dependencies and resource constraints [7]. Currently, the scheduling of DAG tasks in distributed heterogeneous computing environments has become a significant area of research. Topcuoglu et al. [8] introduced the heterogeneous earliest completion time first algorithm (HEFT) algorithm, designed to allocate each subtask to the server that ensures the minimum execution time. He et al. [9] proposed a clustering algorithm based on task replication to generate scheduling. Shen et al. [10] proposed a dependency-aware task offloading and service caching method DTOSC to perform DAG task offloading and service caching in vehicle edge computing. Liu et al. [11] proposed an effective algorithm that jointly considers task scheduling and the on-demand functional configuration of servers to find the optimal task scheduling. In addition to traditional heuristic DAG scheduling methods, list-based scheduling algorithms have been shown to be very effective in reducing Makespan under task-dependent constraints. Hosseini Shirvani and Noorian Talouki [12] proposed a new list scheduling algorithm based on hybrid heuristics, which integrates critical path and priority ranking strategies to achieve efficient task mapping in heterogeneous cloud computing. In order to consider the monetary cost in scheduling, [13] transformed the workflow scheduling problem into a two-objective optimization problem from the perspective of maximum completion time and minimum cost. In addition, Asghari Alaie et al. [14] realizes reliable workflow execution by using a hybrid two-objective discrete cuckoo search algorithm.

In contrast, learning-based algorithms, such as reinforcement learning and deep learning can solve DAG scheduling problems more efficiently. Yan et al. [15] proposed a method based on DRL to learn optimal DAG subtask assignment. In [16], a DAG task offloading method was proposed based on meta-DRL. Goudarzi et al. [17] introduced the actor architecture to the task assignment problem. In [18], a DRL based joint optimization task scheduling algorithm was developed. Geng et al. [19] proposed a multiagent network architecture to schedule tasks. In order to effectively extract dependencies among subtasks, Lee et al. [20] introduced graph convolutional networks (GCN) for DAG task scheduling. In terms of cloud-edge-device collaboration and intelligent scheduling, Liang et al. [21] proposed a dynamic scheduling response mechanism from the perspective of train control. Fan et al. [22] considered resource heterogeneity to conduct hierarchical collaboration on computing resources. Aiming at the load difference problem between vehicles and roadside units, [23] introduced deep reinforcement learning for strategy exploration and resource allocation optimization. Chen et al. [24] proposed a DAG task offloading algorithm named ACE, which uses GCN to capture topological information of DAG subtasks. These task scheduling methods fully consider the dependency between tasks and the resource attributes in the distributed environment, and can achieve a more efficient first scheduling scheme.

However, in the actual process of task scheduling, fault events will inevitably interrupt the execution of tasks. In this case, if all the tasks are rescheduled without considering existing tasks, it will inevitably waste lots of resources and time. To deal with the situation, Lee and Gil [25] proposed a clustering heuristic (CRCH)-based checkpoint and replication to schedule jobs and tolerate faults in the cloud framework. Malik et al. [26] used dynamic standby replication (LSR) strategies to complete the fault scheduling and detection mechanism. Du et al. [27] proposed an approach to effectively discover fault-tolerant strategies. Reference [28] transforms the rescheduling problem into binary nonlinear programming and proposes a task rescheduling algorithm based on particle swarm optimization (PSO). Du et al. [2] proposed both an optimal approach and a heuristic method to maximize migrated users while minimizing latency and deployment costs.

The above research provides many valuable methods for task scheduling and fault recovery, but the heuristic method is suitable for small-scale static scenarios, and it is easy to fall into local optimization in dynamic environment. Although the learning-based method can have better adaptability in the dynamic changing environment, it often has problems, such as high computational complexity and unstable training process. In addition, most of the traditional fault-tolerant scheduling methods focus on static backup or task replication mechanism, ignoring the modeling of fault sensing state and dynamic policy adjustment. To address the aforementioned limitations, a heterogeneous graph neural network-based DAG task rescheduling method for edge computing fault scenarios (EFTR-HGNet) is proposed. Specifically, information is extracted from the heterogeneous state space to achieve 3-D representation learning of task attributes, resource states, and fault features.By dynamically embedding fault features into the heterogeneous graph, our method realizes coordinated representations of task dependencies and resource states, providing

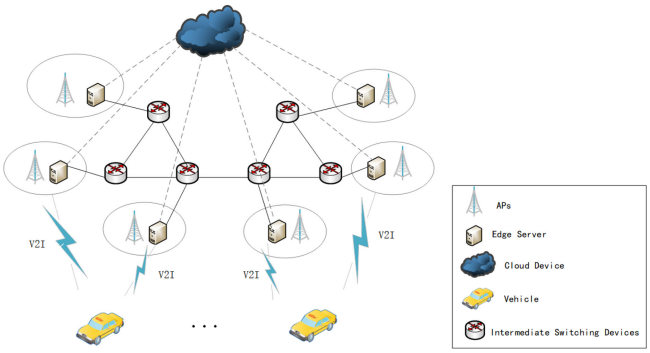

Fig. 1. Edge network architecture.

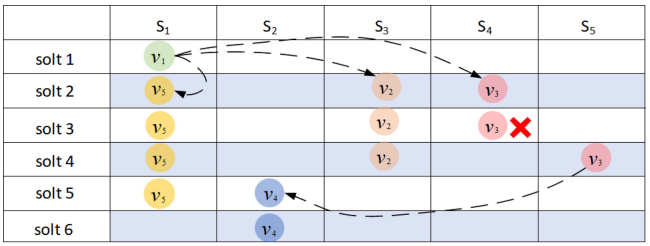

Fig. 2. Time-slot resource matrix.

a more comprehensive state representation for scheduling decisions. Leveraging the online learning advantages of deep reinforcement learning, EFTR-HGNet continuously adapts to environmental changes to achieve adaptive task rescheduling. This approach overcomes the limitations of traditional scheduling methods in fault information fusion and dynamic decision-making, effectively solving the problem of slow response speed during fault recovery.

III. System Model

A. Edge Network Model

The edge network model is a distributed architecture that shifts data processing and task analysis from cloud servers to the edge. Its core components include edge servers, intermediate switching devices, wireless access points (APs), and communication links that interconnect these elements. This architecture is illustrated in Fig. 1.

Edge Servers: Positioned at the network edge, handle the processing and analysis of data collected from nearby sensors or devices.

Intermediate Switching Devices: Responsible for data forwarding and routing, ensure the efficient transmission of information within the network.

Wireless APs: Provide wireless connectivity, enabling user devices to access the edge network.

Communication Links: Facilitate data exchange between the network's components, comprising both wired and wireless connections.

The edge network's topology can be abstracted as $G_{\text{net}} = (N, E)$, where $N = N_S \cup N_D$, N_S is a collection of edge server nodes, $N_S = [n_{S_1}, n_{S_2}, \ldots, n_{S_m}]$, m is the number of edge servers. N_D is the set of intermediate switching devices that perform data forwarding function, $N_D = [n_{D_1}, n_{D_2}, \ldots, n_{D_k}]$, k is the number of switching devices. $e_{ij} \in E$ represents a physical link connecting node n_i and node n_j, $n_{ij} \in N$, forming a communication path. Each server's computing capacity is defined by the number of CPU/GPU cores.

B. Resource Fault Model

We design a centralized matrix representation of resources to describes the allocation of resources and tasks. The timeline is divided into equal-length time slots, represented on the vertical axis, while the system's computing resources are displayed along the horizontal axis. Each slot serves as the smallest unit for resource allocation. A typical time-resource matrix is shown in Fig. 2. The circles represent subtasks from a DAG, and the dashed lines represent dependencies between subtasks, each with a different execution timeline. In time slot t, the number of available cores for resource S_k is represented by $c_k(t)$. Suppose that at slot 3, computing resource S_4 fails, the execution of task υ_3 is interrupted, and the execution of task υ_3 and its subsequent stepchildren υ_4 need to be rescheduled to other available compute resources.

A server in the edge network may fail due to configuration errors or hardware failure [29]. All resources of this server are unavailable during the restart time. Suppose that S_k fails at time t, and take n time slots to recover. If the fault part is unavailable for a long period of time, the service recovery time can be considered infinite, that is, n tends to positive infinity. The size of each time slot is τ, then

$$c_k(t) = \begin{cases} 0, & t \in [t, t + n\tau] \\ b, & \text{otherwise} \end{cases} \tag{1}$$

where $c_k(t)$ represents the number of cores S_k has available in slot t, b represents normal value. The occurrence of a fault event affects some subtasks that are or will be executed and related dependent data. Therefore, the affected tasks must be rescheduled to avoid task failure.

C. Task Computing Model

In the edge network, tasks offloaded by mobile devices can connect to the network at any time through any available wireless AP. A DAG task is represented by $G_{\text{task}} = (V, E)$, G_{task} represents task graph topology. V represents the subtask in G_{task}, $V = [v_1, v_2, \ldots, v_n]$, n represents the number of subtasks. E is the set of edges, representing the direct dependencies of the subtasks. Tasks are assumed to be executed on edge servers located near the wireless APs where the tasks were initially offloaded. The delay costs associated with this process include the following components.

Task Transmission Delay: When a task is offloaded, it will exchange information with AP points, and the information interaction process will generate a certain cost, that is, task transmission delay. Assuming that the communication link between a device and an AP point is set to a flat frequency fast fading Rayleigh channel. According to Shannon's formula, THE transmission delays of the uplink T_{uplink} and downlink T_{downlink} are, respectively, expressed as

$$T_i^{\text{uplink}} = \frac{u_i}{W_i^{ul} \log_2 \left(1 + \frac{p_i^{ul} h_i^{ul}}{N_0 W^{ul}}\right)} \tag{2}$$

$$T_i^{\text{downlink}} = \frac{d_i}{W_i^{dl} \log_2 \left(1 + \frac{p_i^{dl} h_i^{dl}}{N_0 W^{dl}}\right)} \tag{3}$$

where, u_i indicates the amount of data in the uplink, d_i indicates the amount of data in the downlink. h_i^{ul} and h_i^{dl} are channel parameters. N_0 represents noise power spectral density. W^{ul} and W^{dl} are uplink and downlink bandwidth. The transmission power are p_i^{ul} and p_i^{dl}.

Task Execution Delay: It represents the processing time cost of a task at the edge server. The calculation formula can be expressed as

$$T_i^{\text{exe}} = \frac{c_i}{f_j^{\text{edge}}} \tag{4}$$

where, c_i denotes the CPU/GPU cycle required for the edge server computing task υ_i, and f_j^{edge} denotes CPU/GPU computing frequency of edge server S_j.

Therefore, the calculation model of the total delay for task υ_i being offloaded to server S_j includes task transmission delay, task execution delay and task wait time. T_i^{wait} represents the earliest start time of υ_i, that is, when all predecessor subtasks of υ_i have been completed. The calculation formula can be expressed as

$$T_i^{\text{total}} = T_i^{\text{uplink}} + T_i^{\text{exe}} + T_i^{\text{downlink}} + T_i^{\text{wait}}. \tag{5}$$

In an edge service environment, where task scheduling is performed in parallel, the total completion time for scheduling a DAG task is defined as the duration from the start of scheduling to the completion of all subtasks. The formula for calculating this time span is as follows:

$$\text{Makespan} = \max_{i=1,\ldots,n} \{T_i^{\text{total}}\}. \tag{6}$$

Considering the impact of fault events, we must ensure that subtasks affected by fault events can respond quickly to rescheduling. Therefore, the goal of rescheduling is to minimize the task rescheduling completion time while maximizing the rescheduling success rate. The optimization objective is expressed as formula

$$\min \left(\frac{\sum_{\upsilon_i \epsilon V} X_i}{N} \cdot \text{Makespan}\right) \tag{7}$$

$$p_{\min}^{ul} \le p_i^{ul} \le p_{\max}^{ul} \tag{7a}$$

$$p_{\min}^{dl} \le p_i^{dl} \le p_{\max}^{dl} \tag{7b}$$

$$\forall i \in \{1, 2, 3, \ldots, N\} \tag{7c}$$

where, X_i is an variable that indicates whether the task υ_i is successfully executed. If task υ_i fails, $X_i = 1$. N indicates the total number of tasks. $(\sum_{\upsilon_i \epsilon V} X_i / N)$ indicates the task execution failure rate. Equation (7a) and (7b) represent transmit power constraints of mobile devices and edge servers, respectively. Equation (7c) represent the global constraints of the task, ensuring that each node in $\{1, 2, 3, \ldots, N\}$ meets all constraints.

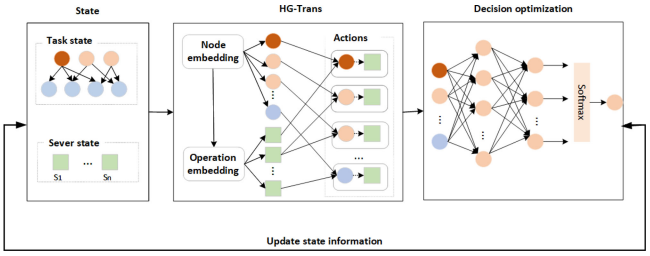

Fig. 3. EFTR-HGNet working frame.

IV. Network Architecture Based on HG-Trans

Task rescheduling is an iterative dynamic decision-making process. At each step, tasks are assigned to available computing resources until all tasks are scheduled. Fault tolerance is introduced in task scheduling and decision making. The mechanism interacts with reinforcement learning agents through state representation and state transfer. When a resource failure is detected, the state transition automatically reflects the affected task to the next state state_{t+1}. After the agent observes this updated state containing fault information at $t+1$, the system will then determine subsequent actions based on this updated state representation, dynamically reassigning interrupted tasks to optimal compute nodes through its adaptive scheduling policy. The working framework of the proposed EFTR-HGNet method is shown in Fig. 3. The detailed workflow is as follows.

1) *State Representation:* At each scheduling iteration, the system first captures the current scheduling environment and resource states, constructing a heterogeneous graph representation of the task-server relationships.
2) *Heterogeneous Graph Embedding:* The constructed graph is then processed by the HG-Trans through a clearly defined two-stage embedding process.
   a) *Node Embedding:* Extract and encode the local features of task nodes and server nodes, and capture their individual attributes and structural roles in heterogeneous graphs.
   b) *Operation Embedding:* Aggregate and integrate the node embeddings to generate a unified global representation, reflecting the overall scheduling status and task-resource relationships.
   c) *Action Generation:* The embedded features are subsequently fed into the decision network to generate an action probability distribution.
   d) *Decision Execution:* Scheduling actions are sampled from this distribution, determining task assignment and resource allocation.
   e) *State Update and Feedback:* Once actions are executed, the environment transitions into a new state, restarting the iteration from step (1).

A. Task Scheduling Decision-Making Process

The dynamic task scheduling is modeled as a Markov decision process (MDP). Through the agent's continuous interaction with the state environment, the appropriate

available computing resources are selected. The decision process of task scheduling can be defined as follows.

State: The combined reflection of all task schedules and computing resource states at step t constitutes the state$_t$. We construct a simplified model that approximates the task information at state$_t$ as a set of dependency information about the running task and the ready task $D(t) = \{(v_i, v_j)|, i, j \in \{1, 2, \ldots, N\}\}$ where v_j is the direct predecessor of v_i. The resource status of S_k is represented by a vector $R_k(t) = \{(r_k, a_k(t), c_k(t))|k \in \{1, 2, \ldots, M\}\}$, where r_k represents the type of computing resource, $a_k(t)$ indicates the availability status of the resource at time t, $c_k(t)$ represents the remaining computing capacity at time t, specifically the number of available CPU/GPU cores. In the state representation, we introduce the state variable $F(t) = \{(S_k, f_k)|S_k \in S, f_k \in \{0, 1\}\}$ to represent fault state. $f_k = 1$ indicates that the server is faulty, and $f_k = 0$ indicates that the server is normal. If the server to which the task is bound is faulty, the task enters the interrupted state. In state$_t$, we recursively compute the execution time of all scheduling operations from start to finish, denoted as $T(t)$. If the direct predecessor v_j of task v_i has already been scheduled, then the actual completion time of v_i is the sum of its execution time and the actual completion time of v_j, i.e., $T_i(t) = T_i^{\text{exe}} + T_j(t-1)$. Therefore, the state set at time step t can be represented as

$$S = \left\{\text{state}_t = \left(D(t), R_k(t), F(t), T(t)\right)|t \in T\right\}. \tag{8}$$

Action: In the definition of action, we consider task selection and resource allocation as a compound decision. The action action$_t \in A_t$ is defined as a V-S pair *(V, S)* at time t. where $v_i \in V, S_k \in S$. If S_k is idle, v_i can be executed directly in S_k. Since each subtask can select up to one resource at a time, therefore $|A_t| \leq n \times m$. n is the total number of subtasks, m is the total number of computing resources. When a server resource is available, the action set is the task that is selected to run on this computing resource. When a server resource is unavailable due to a failure, the RL agent quickly responds to the task being executed on the failing server by putting it back into the pending queue and clearing all computing resource allocation information related to the task on the failed resource. Therefore, the action set at time step can be represented as

$$A_t = \{\text{action}_t = (v_i, S_k)|v_i \in V, S_k \in S\}. \tag{9}$$

Transition: The transfer function represents the probability that action action$_t$ transitions from state state$_t$ to state state$_{t+1}$. In the process of state transition, fault tolerance and adaptive adjustment mechanisms are introduced, which are formalized as

$$P_i(t+1) = \begin{cases} \{(v_i, S_k)|t\}, & f_k = 1 \&\& v_i \text{ on } S_k \\ \left\{\left(v_i, S_j\right)|t\right\}, & \text{otherwise} \end{cases} \tag{10}$$

where, S_k indicates the faulty server and S_j indicates another available server.

Reward: At each time t, rewrard$_t$ is rewarded for the optimization goal when the RL agent providing a rescheduling policy. We targeted Makespan for optimization, and the smaller the value, the greater the reward

$$R(\text{Makespan}) = e^{-\text{Makespan}(t)}. \tag{11}$$

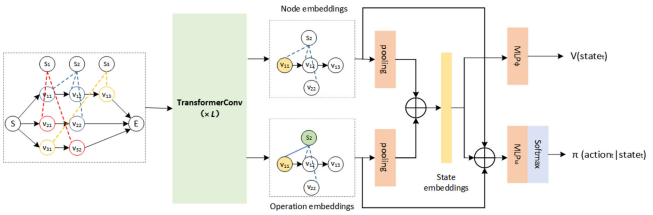

Fig. 4. HG-trans network architecture.

The RL agent performs actions based on the state accessed and the current policy, interacting with the problem to be solved and gradually adjusting the policy to optimize the objective function.

B. HG-Trans Network Architecture

We define a new heterogeneous graph structure $H_t = (V, S, \varepsilon_t)$ to represent the complex relationship between tasks and servers in the scheduling state, where V is the set of task node, S is the set of server node, ε_t represents the arc set of V-S, each task node corresponds to a server. When action (v_i, S_k) is taken at t, only $e_{ik} \in \varepsilon_t$ is retained, and the other V-S arcs of v_i are removed to obtain H_{t+1}.

When dealing with heterogeneous graphs, most of the traditional models based on neighborhood aggregation can only transfer information in a local scope, and can not fully reflect the characteristics of the global interaction between tasks and servers. In order to effectively extract information from heterogeneous graphs, we introduce a heterogeneous graph neural network based on Transformer. Then a two-stage embedding process is adopted to map the nodes in heterogeneous graphs into D-dimensional embedding, considering the graph topology and node characteristics. The first stage updates the node embedding, and the second stage updates the operation embedding. The network architecture is shown in Fig. 4. Heterogeneous graphs contain different types of nodes (tasks and servers) and edges (V-S). In order to distinguish these heterogeneous information, the model introduces special type coding for each node and edge, so that the information aggregation strategy can be automatically adjusted according to the type of node and edge in self-attention computation. By stacking multiple Transformer layers, the model can aggregate information layer by layer, effectively capturing everything from initial local features to global state. Our two-stage embedding is based on this hierarchical information integration. The first stage focuses on local structure and node characteristics, and the second stage further integrates the influence of scheduling operations to form a more decision-guiding embedded representation.

Node Embedding: The graph convolution layer updates the embedding of each node by aggregating the information of neighbor nodes, thus realizing the feature extraction of graph data. We design a TransformerConv network with self-attention to aggregate nodes and their neighbors. First, the input node features are transformed linearly, and then the attention weight of each node to its neighbor nodes is calculated. The attention coefficient A_{ij} between node v_i and

every node υ_j in the first-order field $\mathcal{N}(i)$ is

$$A_{ij} = \frac{\left(Q_i \cdot K_j^T\right)}{\sqrt{d_k}} \tag{12}$$

where A_{ij} is the attention weight of node υ_i to node υ_j. Q_i is the query vector. K_j is the key vector. d_k is the dimension of the key vector and is used to scale the dot product result to avoid large values. The coefficients are normalized by the Softmax function in the neighborhood

$$\alpha_{ij} = \frac{\exp\left(A_{ij}\right)}{\sum_{q\in N(i)} \exp\left(A_{iq}\right)} \quad \forall j \in \mathcal{N}(i). \tag{13}$$

In order to update the features of the nodes, the information of the neighbor nodes is weighted and aggregated using attention weights

$$H_i^{(l+1)} = \sigma\left(\sum_{j\epsilon\mathcal{N}(i)} \alpha_{ij} V_j\right) \tag{14}$$

where $H_i^{(l+1)}$ is the feature representation of node υ_i at layer l, and σ is the nonlinear activation function. V_j is embedded representation of node υ_j. The aggregated information is nonlinear transformed through the feedforward network

$$H_i^{(l+1)} = \mathrm{ReLU}\left(W_f \cdot H_i^{(l+1)} + b_f\right) \tag{15}$$

where, W_f and b_f, respectively, represent the weight and bias parameters of the linear transformation, and finally overlay the multilayer Transformer update task and server node embedding.

Operation Embedding: The node embedding generated in the first stage is extracted by multilayer perceptron (MLP) and the probability distribution of the actions is generated. First, all node embedding are summarized by means pooling, and the state value $V(\text{state}_t)$ is estimated by 1-D projection. Assuming that the projection matrix is $W_v \in R^{d\times 1}$, according to the formula

$$V = W_v^T \cdot \frac{1}{N}\sum_{i=1}^{N} h_i \tag{16}$$

where h_i represents the eigenvector of the ith node. $(1/N)\sum_{i=1}^{N} h_i$ is the global eigenvector after mean pooling. The embedding of the available tasks are then aggregated into a batch matrix, where each row corresponds to an embedding vector of a task node

$$H_T = [h_1, h_2, \ldots, h_M]^T \in R^{M\times d}. \tag{17}$$

Aggregate the embedding representations of the currently available tasks, and then map these embedding representations into a 1-D vector space to generate a score for each task. The probability distribution of the generated action is normalized by the Softmax function of the fully connected layer

$$\pi_i = \frac{\exp(o_i)}{\sum_j \exp\left(o_j\right)} \tag{18}$$

where π_i is the probability of the ith action and o_i is the ith component in the output vector o.

C. Architecture of the RL Agent

To optimize decision-making, we enhance the two neural networks used in the Actor-Critic (A2C) algorithm.

Policy Network (Actor): Represented as $\pi_\theta(\text{action}_\text{t}|\text{state}_\text{t})$ with parameters θ, it generates a probability distribution over possible actions.

Value Network (Critic): Parameterized by θ_ϑ, it estimates the value of a state $V_{\theta_\vartheta}(\text{state}_t)$, which reflects the desirability of being in state state_t to achieve the optimization goal.

The state value $V(\text{state})$ measures how beneficial a given state state_t is for maximizing the objective function. We optimize the value function by minimizing the gap between the state-value function or action-value function estimated by the Bellman equation and the actual value. At the same time, the cumulative reward maximization problem is transformed into a joint optimization problem of strategy network gradient and dominance function. That is

$$\nabla_\theta J(\theta) = \mathbb{E}_t\left[\nabla_\theta \log \pi_\theta(\text{action}_t|\text{state}_t)\cdot A(\text{state}_t, \text{action}_t)\right]. \tag{19}$$

Here, $A(\text{state}_t, \text{action}_t)$ stands for advantage function.It is used to measure the quality of action action_t relative to the average performance of the current strategy, and then guide the optimization direction of the strategy. The advantage A is derived using the current estimates of the quadruple $(\text{state}_t, \text{action}_t, \text{reward}_t, \text{state}_{t+1})$ and the value function V. In the training process, as the strategy gradually converges, the strategy will be more inclined to deterministically select the action with higher reward, resulting in premature convergence or falling into the local optimal solution. In order to encourage the strategy to maintain some exploration capability, we add entropy to the objective function. So the final objective function is

$$\begin{aligned}\nabla_\theta J(\theta) = {} & \mathbb{E}_t\left[\nabla_\theta \log \pi_\theta(\text{action}_t \mid \text{state}_t)\cdot A(\text{state}_t, \text{action}_t)\right] \\ & + \beta\mathbb{E}_t[\nabla_\theta \mathbb{H}(\pi_\theta(\cdot \mid \text{state}_t))]\end{aligned} \tag{20}$$

where $\mathbb{H}(\pi_\theta(\cdot|\text{state}_t))$ represents the entropy function of the probability distribution, and β is the hyperparameter that controls the effect of entropy regularization. RL agents update the Critic and Actor networks by constantly interacting with the environment, collecting data and computing dominance functions. Ultimately, the strategy is optimized so that the agent gets the maximum long-term reward for a given task. In each round of training, the agent will adjust its strategy so that in the same state, the selected action will gradually become more reasonable, so as to gradually approach the optimal strategy.

D. Algorithm

We designed an edge-fault scenario task rescheduling method based on heterogeneous graph neural networks (EFTR-HGNet), as shown in Algorithm 1.

In the algorithm, we initialize the Actor-Critic network with the given parameters. For each turn, a batch containing B instances is sampled from the environment, and for each instance in the batch, the simulation is run until the termination

Algorithm 1 EFTR-HGNet Algorithm

1: **Input:** Environment *env*, Actor-Critic network with Transformer network. Trainable parameters θ and ϑ, Number of episodes N, batch size B.
2: **Output:** Actor network $\pi_\theta(\text{action}_t|\text{state}_t)$, Critic network $V_\vartheta(state_t)$ and Makespan of task rescheduling.
3: Initialize network with parameters θ and ϑ
4: **for** episode $= 1, 2, \ldots, N$ **do**
5: **for** $b = 1, 2, \ldots, B$ **do**
6: Initialize $state_t$ based on instance b_i
7: **while** $state_t$ is not terminal **do**
8: $z_t = Transformer(state_t)$
9: Sample action $action_t \sim \pi_\theta(\cdot|state_t)$
10: $reward_t, state_{t+1} \leftarrow (state_t, action_t)$
11: Score $(state_t, action_t, reward_t, state_{t+1})$
12: **end while**
13: **end for**
14: $A_t = reward_t + \gamma V_\vartheta(state_{t+1}) - V_\vartheta(state_t)$
15: $V_t^{target} = reward_t + \gamma V_\vartheta(state_{t+1})$
16: Compute the A2C loss $\mathcal{L}$
17: $\theta, \omega \leftarrow Optimizer(\mathcal{L}, \theta, \vartheta)$
18: Update network parameters
19: **if** episode mod $m = 0$ **then**
20: Validate the policy
21: **end if**
22: **if** episode mod $n = 0$ **then**
23: Simple a new batch of B instances
24: **end if**
25: **end for**
26: **return**

state is reached. Line 7–12 extract the embedding vector through the Transformer network and sample actions, store state transitions, and rewards based on the current policy. Line 14 and 17 use Critic networks to calculate strengths and value objectives, calculate A2C losses and optimize parameters for multiple iterations. Repeat the above process until the predetermined training rounds are reached, and return to the trained network. Finally, output the optimal action set of rescheduling decision and Makespan. The total time complexity is estimated as $O(n(n+m)^2 d))$. n indicates the number of task nodes, m indicates the number of server resources, and d indicates the Transformer embedding dimension.

V. Experiments

A. Experimental Settings

To verify the effectiveness of our model, we have developed an edge scheduling simulation environment. The software environment is Python. The core algorithm was developed on the PyTorch 1.12.1 framework and ran on a workstation configured with an Intel i7-11700 CPU and 16 GB of memory.

In this section, we construct a simulation environment to simulate the task rescheduling process of edge server failure in a harsh environment. We set up some edge servers and each server is distributed on different work surfaces to support various tasks. Each edge server is configured with limited computing resources, including CPUs and GPUs. It can handle different types of tasks. Communication between servers is implemented through a wireless network.We randomly generate DAG tasks with different topologies in the simulator, and each subtask corresponds to different task attributes and resource requirements. Specific simulation parameters are given in Table I.

TABLE I
Parameter for Simulation

Parameters	Values
The number of task type	4
Type of computing resources for each server S_k	CPU or GPU
The number of CPU/GPU cores of each server S_k	[1,5]
Processing capacity of server S_k	[1 GHz, 5 GHz]
Bandwidth of links $e_{i,j}$	[200 Mbps, 500 Mbps]
Number of sub-tasks in a DAG	[10,100]
The amount of data transferred by a task	[10MB, 20MB]
CPU/GPU requirements of each sub-task	[0.1G, 0.3G]
Time slot length τ	1ms
Time slots requirements for failure recovery	10ms

In task scheduling system, Makespan, speedup and schedule length ratio (SLR) are three key performance evaluation indexes, which measure the efficiency and effect of scheduling system from different angles [30].

Makespan refers to the maximum time required to complete all tasks, usually the time elapsed from the start of scheduling until all tasks are completed. If Makespan is large, it means that resource allocation during task scheduling is not optimized enough, some processor resources may be idle, or tasks may be delayed due to resource conflicts. Therefore, minimizing Makespan is often a major goal of optimizing scheduling algorithms.

Scheduling length ratio (SLR) is the ratio of the actual schedule completion time to the ideal schedule completion time. The ideal scheduling completion time refers to the time when the task should be completed without any resource conflict or scheduling delay. The calculation formula is as follows:

$$\text{SLR} = \frac{\text{Makespan}}{\sum_{p_i \epsilon P_{\min}} \min_{q_j \epsilon Q} e_{i,j}}. \tag{21}$$

The closer SLR is to 1, the smaller the gap between the actual scheduling scheme and the ideal scheme, and the higher the efficiency of the scheduling system [31]. The lower SLR value indicates that the scheduling system can approach the optimal ideal scheduling scheme well without too much delay or waste of resources.

Speedup is used to measure the parallelization effect of scheduling algorithms in multicore or distributed environments. It reflects the performance improvement of multicore processing or distributed computing over serial computing. The calculation formula is as follows:

$$\text{Speedup} = \frac{\min_{q_j \epsilon Q} \left\{ \sum_{p_i \epsilon V} e_{i,j} \right\}}{\text{Makespan}} \tag{22}$$

when the Speedup value is high, it indicates that task scheduling can efficiently distribute tasks to multiple processors. Tasks

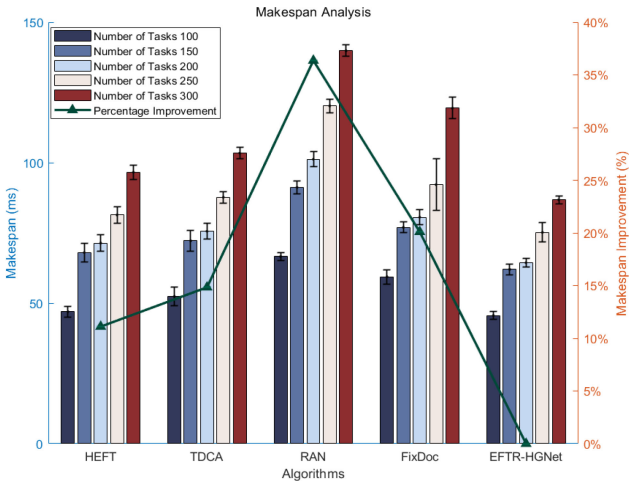

Fig. 5. Performance and improvement comparison based on Makespan.

are more evenly distributed among processors, significantly reducing overall execution time.

We use the following four scheduling algorithms as comparative baselines to evaluate the proposed task rescheduling algorithm (EFTR-HGNet).

1) *Based on HEFT [8]:* It is an efficient heuristic scheduling algorithm in nonrepetitive and batch processing mode.
2) *Task Replication And Cluster-Based Scheduling Algorithm (TDCA) [9]:* Use task replication and clustering techniques to minimize task completion time in batch mode.
3) *Scheduling Algorithm Based on Task Replication and Greedy Policy (FixDoc) [11]:* By copying the precursor task to multiple processors, the task can be directly executed locally and select the current optimal solution, eventually approaching the global optimal solution.
4) Random policy-based Task Scheduling algorithm (RAN): Randomly select the computing resources of the execution server for each subtask to schedule.

B. Experimental Results and Analysis

In the EFTR-HGNet method proposed in this article, several key hyperparameters are involved, including the learning rate lr, embedding dimension d, entropy regularization coefficient β, and batch size B. Currently, these hyperparameters have been determined through extensive experimentation, with specific parameter configurations as follows. $lr = 1 \times 10^{-3}$, $d = 128$, $\beta = 0.01$ and $B = 32$.

Experiment 1 aims to compare the variations in the maximum completion time (Makespan) of EFTR-HGNet and four baseline algorithms as the number of tasks N increases in the range $[100, 150, 200, 250, 300]$. The experimental results are shown in Fig. 5. As can be observed from the figure, with the increase in task quantity, the Makespan of all algorithms shows an upward trend. However, the growth rate of EFTR-HGNet is noticeably slower, and its overall performance consistently outperforms the other baseline methods. In particular, the Makespan of FixDoc and RAN is significantly higher.

TABLE II
PERFORMANCE AND IMPROVEMENT COMPARISON BASED ON SLR

Number of tasks		EFTR-HGNet	HEFT	TDCA	FixDoc	RAN
100	SLR_{ave}	1.31	1.42	1.39	1.48	1.54
	Improvement rate	-	7.75% ↓	5.76% ↓	11.49% ↓	14.94% ↓
150	SLR_{ave}	1.44	1.47	1.49	1.52	1.58
	Improvement rate	-	2.04%↓	3.36% ↓	5.26%↓	8.86% ↓
200	SLR_{ave}	1.26	1.37	1.46	1.57	1.62
	Improvement rate	-	8.03% ↓	13.70% ↓	19.75% ↓	22.22% ↓
250	SLR_{ave}	1.41	1.48	1.52	1.55	1.67
	Improvement rate	-	4.73%↓	7.24% ↓	9.03%↓	15.57%↓
300	SLR_{ave}	1.54	1.61	1.69	1.72	1.86
	Improvement rate	-	4.35%↓	8.88% ↓	10.47%↓	17.20% ↓

This is because FixDoc allows each subtask to be executed concurrently on multiple computing nodes, which introduces redundancy and prolongs task completion time. On the other hand, RAN suffers from poor scheduling efficiency due to its inherent randomness and lack of strategy. We enhance the statistical reliability of the experimental conclusions by plotting 95% confidence interval (CI) error bars for all bars in the figure, calculated from multiple independent experiments using different random seeds. It can be seen that EFTR-HGNet consistently maintains a narrow CI range across all task scales, indicating that it achieves more stable and robust scheduling results. When the number of tasks $N = 200$, the maximum completion time of EFTR-HGNet algorithm is increased by 11.11%, 14.67%, 37.25% and 20.99%, respectively, compared with HEFT, TDCA, RAN, and FixDoc. Therefore, under the same fault condition, our proposed EFTR-HGNet algorithm can balance short-term benefits and long-term rewards, maximize the overall scheduling performance through policy optimization and show higher execution efficiency in rescheduling.

Experiment 2 was used to compare the SLR of EFTR-HGNet algorithm with the other four baseline algorithms as the number of tasks changes. The experimental results are shown in Table II. In Table II, the SLR value of EFTR-HGNet is all between $[1, 1.6]$, while the SLR value of other algorithms is higher than the SLR value of EFTR-HGNet, indicating that EFTR-HGNet algorithm is closer to the ideal scheduling length. When $N = 200$, the SLR of EFTR-HGNet algorithm increased by 8.03%, 13.70%, 19.75% and 22.22%, respectively, compared with HEFT, TDCA, FixDoc and RAN. Therefore, under the same fault condition, our proposed EFTR-HGNet algorithm performs better in rescheduling algorithm performance and is closer to the ideal condition.

Experiment 3 was used to compare Speedup of EFTR-HGNet algorithm with the other four baseline algorithms as the number of tasks changes. The experimental results are shown in Table III. In Table III, the Speedup value of EFTR-HGNet is significantly higher than that of other algorithms. When the number of tasks $N = 200$, the Speedup of EFTR-HGNet algorithm increased by 36.5%, 39.94%, 28.99% and 44.01% compared with HEFT, TDCA, FixDoc and RAN, respectively. Therefore, EFTR-DRL algorithm has a higher degree of task parallelization and higher resource utilization.

TABLE III
PERFORMANCE AND IMPROVEMENT COMPARISON BASED ON SPEEDUP

Number of tasks		EFTR-HGNet	HEFT	TDCA	FixDoc	RAN
100	$Speedup_{ave}$	2.89	2.83	2.76	2.78	2.64
	Improvement rate	-	2.12% ↑	4.71% ↑	3.96% ↑	9.50% ↑
150	$Speedup_{ave}$	3.02	2.92	2.85	2.82	2.75
	Improvement rate	-	3.42% ↑	5.96% ↑	7.09% ↑	9.82% ↑
200	$Speedup_{ave}$	4.45	3.26	3.18	3.45	3.09
	Improvement rate	-	36.5% ↑	39.94% ↑	28.99% ↑	44.01% ↑
250	$Speedup_{ave}$	4.41	3.42	3.21	3.57	3.32
	Improvement rate	-	28.95% ↑	37.38% ↑	23.53% ↑	32.83% ↑
300	$Speedup_{ave}$	4.36	3.37	3.43	3.68	3.39
	Improvement rate	-	29.38% ↑	27.11% ↑	18.48% ↑	28.61% ↑

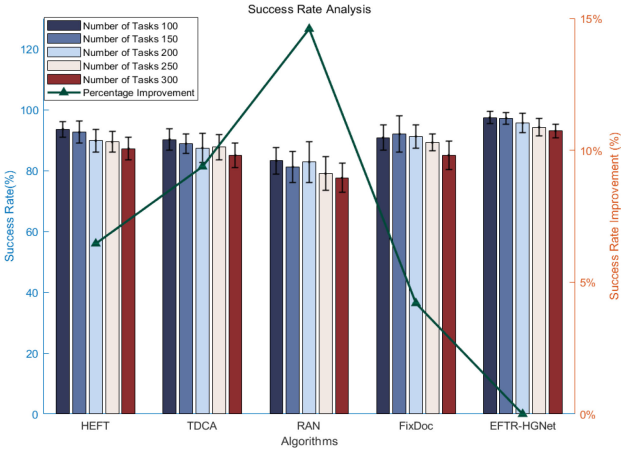

Fig. 6. Performance and improvement comparison based on success rate.

Experiment 4 compares the task success rate of the EFTR-HGNet algorithm with four baseline algorithms under varying task quantities. We set the maximum response delay threshold to 5*ms*. If a task's waiting time exceeds this threshold, it is considered a failure. The experimental results are shown in Fig. 6. As observed in the figure, the task success rate of EFTR-HGNet consistently falls within the range of 90% to 100%, outperforming all baseline algorithms across all task scales. When the number of tasks $N = 200$, the average task success rate of EFTR-HGNet improves by 6.3%, 9.4%, 14.6%, and 4.2% compared to HEFT, TDCA, RAN, and FixDoc. The results demonstrate that EFTR-HGNet achieves statistically significant performance superiority compared to baseline algorithms, eliminating stochastic fluctuations as a contributing factor. Consequently, the framework maintains operational efficiency in task failure scenarios under identical fault conditions while optimizing task success rates.

Experiment 5 analyzed the performance of different algorithms on task response delay by comparing the cumulative distribution function (CDF) on response delay under the condition of the same number of tasks. The experimental results are shown in Fig. 7. The horizontal coordinate represents the delay time (in milliseconds) of the task response, and the vertical coordinate represents the proportion of tasks that have been completed within a certain delay time. When the CDF curve is more left and steeper, it is able to complete

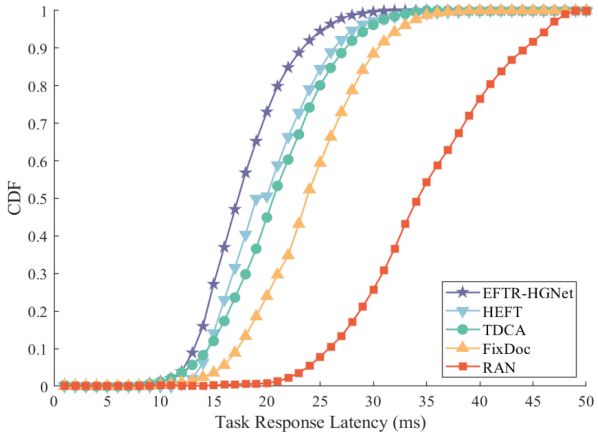

Fig. 7. CDF of task response delay of different algorithms.

more tasks in less time, and the performance is better. As can be seen from the figure, the CDF curve of EFTR-HGNet is farthest to the left and the curve slope is the largest, which indicates that it completes most tasks in the shortest delay time. At around 20*ms*, more than 90% of the task had been completed. Compared with other algorithms, EFTR-HGNet shows obvious advantages in different task delay periods.

We evaluate the impact of task quantity on rescheduling performance, and results show that EFTR-HGNet consistently outperforms baselines across multiple metrics and maintains strong environmental adaptability and robustness under diverse and dynamic edge computing conditions.

VI. CONCLUSION AND FUTURE DIRECTION

This article presents a DAG task rescheduling method (EFTR-HGNet) based on heterogeneous graph neural networks for edge computing in fault scenarios to minimize the impact of failures with intelligent decision making. In the process of task scheduling and decision-making, heterogeneous graph structures is used for state representation and a fault-tolerant mechanism is introduced to achieve adaptive task migration under resource failures. By constructing a neural network based on HG-Trans and adopting the A2C algorithm to optimize the strategy, efficient task resscheduling decisions is achieved. Experimental results show that compared with other baseline algorithms, the response delay of the task is significantly reduced. The current method is specifically designed for Internet of Vehicles scenarios characterized by three core challenges: time-sensitive vehicular computing tasks requiring low-latency processing, heterogeneous resource distribution across network nodes, and dynamic environments prone to frequent system failures. Both the task scheduling mechanisms and system state representations have been fundamentally architected around the unique operational constraints and topological features inherent to vehicular network system. In future work, we will consider the model's adaptability in complex scenarios, such as concurrent failures of multiple nodes, Byzantine failures, and intermittent network interruptions.

REFERENCES

[1] D. Xu, "Device Scheduling and computation offloading in mobile edge computing networks: A novel NOMA scheme," *IEEE Trans. Veh. Technol.*, vol. 73, no. 6, pp. 9071–9076, Jun. 2024, doi: 10.1109/TVT.2024.3352262.

[2] W. Du, Q. He, Y. Ji, C. Cai, and X. Zhao, "Optimal user migration upon server failures in edge computing environment," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, 2021, pp. 272–281, doi: 10.1109/ICWS53863.2021.00045.

[3] P. Suresh et al., "Optimized task scheduling approach with fault tolerant load balancing using multi-objective cat swarm optimization for multi-cloud environment," *Appl. Soft Comput.*, vol. 165, Nov. 2024, Art. no. 112129, doi: 10.1016/j.asoc.2024.112129.

[4] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multihop offloading of multiple DAG tasks in collaborative edge computing," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4893–4905, Mar. 2021, doi: 10.1109/JIOT.2020.3030926.

[5] Z. Liu, M. Liwang, S. Hosseinalipour, H. Dai, Z. Gao, and L. Huang, "RFID: Towards low latency and reliable DAG task scheduling over dynamic vehicular clouds," *IEEE Trans. Veh. Technol.*, vol. 72, no. 9, pp. 12139–12153, Sep. 2023, doi: 10.1109/TVT.2023.3266582.

[6] C. Shyalika, T. Silva, and A. Karunananda, "Reinforcement learning in dynamic task scheduling: A review," *SN Comput. Sci.*, vol. 1, p. 306, Sep. 2020, doi: 10.1007/s42979-020-00326-5.

[7] B. Sun et al., "Edge generation scheduling for DAG tasks using deep reinforcement learning," *IEEE Trans. Comput.*, vol. 73, no. 4, pp. 1034–1047, Apr. 2024, doi: 10.1109/TC.2024.3350243.

[8] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002, doi: 10.1109/71.993206.

[9] K. He, X. Meng, Z. Pan, L. Yuan, and P. Zhou, "A novel task-duplication based clustering algorithm for heterogeneous computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 2–14, Jan. 2019, doi: 10.1109/TPDS.2018.2851221.

[10] Q. Shen, B.-J. Hu, and E. Xia, "Dependency-aware task offloading and service caching in vehicular edge computing," *IEEE Trans. Veh. Technol.*, vol. 71, no. 12, pp. 13182–13197, Dec. 2022, doi: 10.1109/TVT.2022.3196544.

[11] L. Liu, H. Tan, S. H.-C. Jiang, Z. Han, X.-Y. Li, and H. Huang, "Dependent task placement and scheduling with function configuration in edge computing," in *Proc. Int. Symp. Quality Service (IWQoS)*, 2019, pp. 1–10, doi: 10.1145/3326285.3329055.

[12] 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, doi: 10.1016/j.parco.2021.102828.

[13] M. Hosseini Shirvani and R. Noorian Talouki, "Bi-objective scheduling algorithm for scientific workflows on cloud computing platform with makespan and monetary cost minimization approach," *Complex Intell. Syst.*, vol. 8, no. 2, pp. 1085–1114, Apr. 2022, doi: 10.1007/s40747-021-00528-1.

[14] Y. Asghari Alaie, M. Hosseini Shirvani, and A. M. Rahmani, "A hybrid bi-objective scheduling algorithm for execution of scientific workflows on cloud platforms with execution time and reliability approach," *J. Supercomput.*, vol. 79, no. 2, pp. 1451–1503, Feb. 2023, doi: 10.1007/s11227-022-04703-0.

[15] 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 Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5404–5419, Aug. 2020, doi: 10.1109/TWC.2020.2993071.

[16] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, Jan. 2021, doi: 10.1109/TPDS.2020.3014896.

[17] M. Goudarzi, M. Palaniswami, and R. Buyya, "A distributed deep reinforcement learning technique for application placement in edge and fog computing environments," *IEEE Trans. Mobile Comput.*, vol. 22, no. 5, pp. 2491–2505, May 2023, doi: 10.1109/TMC.2021.3123165.

[18] X. Wei, L. Cai, N. Wei, P. Zou, J. Zhang, and S. Subramaniam, "Joint UAV trajectory planning, DAG task scheduling, and service function deployment based on DRL in UAV-empowered edge computing," *IEEE Internet Things J.*, vol. 10, no. 14, pp. 12826–12838, Jul. 2023, doi: 10.1109/JIOT.2023.3257291.

[19] L. Geng, H. Zhao, J. Wang, A. Kaushik, S. Yuan, and W. Feng, "Deep-reinforcement-learning-based distributed computation offloading in vehicular edge computing networks," *IEEE Internet Things J.*, vol. 10, no. 14, pp. 12416–12433, Jul. 2023, doi: 10.1109/JIOT.2023.3247013.

[20] H. Lee, S. Cho, Y. Jang, J. Lee, and H. Woo, "A global DAG task scheduler using deep reinforcement learning and graph convolution network," *IEEE Access*, vol. 9, pp. 158548–158561, 2021, doi: 10.1109/ACCESS.2021.3130407.

[21] H. Liang, L. Zhu, F. R. Yu, and C. Yuen, "Cloud-edge-end collaboration for intelligent train regulation optimization in TACS," *IEEE Trans. Veh. Technol.*, vol. 74, no. 1, pp. 454–465, Jan. 2025, doi: 10.1109/TVT.2024.3462708.

[22] W. Fan, P. Chen, X. Chun, and Y. Liu, "MADRL-based model partitioning, aggregation control, and resource allocation for cloud-edge-device collaborative split federated learning," *IEEE Trans. Mobile Comput.*, vol. 24, no. 6, pp. 5324–5341, Jun. 2025, doi: 10.1109/TMC.2025.3530482.

[23] W. Fan, Y. Zhang, G. Zhou, and Y. Liu, "Deep reinforcement learning-based task offloading for vehicular edge computing with flexible RSU-RSU cooperation," *IEEE Trans. Intell. Transp. Syst.*, vol. 25, no. 7, pp. 7712–7725, Jul. 2024, doi: 10.1109/TITS.2024.3349546.

[24] J. Chen, Y. Yang, C. Wang, H. Zhang, C. Qiu, and X. Wang, "Multitask offloading strategy optimization based on directed acyclic graphs for edge computing," *IEEE Internet Things J.*, vol. 9, no. 12, pp. 9367–9378, Jun. 2022, doi: 10.1109/JIOT.2021.3110412.

[25] J. Lee and J. Gil, "Adaptive fault-tolerant scheduling strategies for mobile cloud computing," *J. Supercomput.*, vol. 75, no. 8, pp. 4472–4488, Aug. 2019, doi: 10.1007/s11227-019-02745-5.

[26] M. K. Malik, A. Singh, and A. Swaroop, "A planned scheduling process of cloud computing by an effective job allocation and fault-tolerant mechanism," *J. Ambient Intell. Human. Comput.*, vol. 13, no. 2, pp. 1153–1171, Feb. 2022, doi: 10.1007/s12652-021-03537-7.

[27] W. Du et al., "Fault-tolerating edge computing with server redundancy based on a variant of group degree centrality," in *Proc. Int. Conf. Service-Orient. Comput. (ICSOC)*, 2020, pp. 198–214, doi: 10.1007/978-3-030-65310-1_16.

[28] K. Shao, H. Fu, Y. Song, and B. Wang, "A PSO improved with imbalanced mutation and task rescheduling for task offloading in end-edge-cloud computing," *Comput. Syst. Sci. Eng.*, vol. 47, no. 2, pp. 2259–2274, 2023, doi: 10.32604/csse.2023.041454.

[29] Y. Kanizo, O. Rottenstreich, I. Segall, and J. Yallouz, "Optimizing virtual backup allocation for middleboxes," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2759–2772, Oct. 2017, doi: 10.1109/TNET.2017.2703080.

[30] Y. Cheng, Z. Wu, K. Liu, Q. Wu, and Y. Wang, "Smart DAG tasks scheduling between trusted and untrusted entities using the MCTS method," *Sustainability*, vol. 11, no. 7, p. 1826, 2019, doi: 10.3390/su11071826.

[31] Q. Wu, Z. Wu, Y. Zhuang, and Y. Cheng, "Adaptive DAG tasks scheduling with deep reinforcement learning," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.*, 2018, pp. 477–490, doi: 10.1007/978-3-030-05054-2-37.

Lu Yao is currently pursuing the master's degree in software engineering with Inner Mongolia University, Hohhot, China.

Her research interests focus on software reliability engineering for mobile edge computing (MEC) systems in distributed computing environments.

Yan Wang (Member, IEEE) received the Ph.D. degree in computer science from Inner Mongolia University, Hohhot, China, in 2015.

She is currently an Associate Professor and a Master Supervisor of Computer Science and Technology with Inner Mongolia University. She has published more than 40 papers in international conferences and journals in her field. Her research interests include service computing, formal methods, and software technology.

Keqin Li (Fellow, IEEE) received the B.S. degree in computer science from Tsinghua University, Beijing, China, in 1985, and the Ph.D. degree in computer science from the University of Houston, Houston, TX, USA, in 1990.

He is a SUNY Distinguished Professor with the State University of New York, New Paltz, NY, USA, and a National Distinguished Professor with Hunan University, Changsha, China. He has authored or co-authored more than 1130 journal articles, book chapters, and refereed conference papers. He holds nearly 80 patents announced or authorized by the Chinese National Intellectual Property Administration.

Dr. Li received the IEEE TCCLD Research Impact Award from the IEEE CS Technical Committee on Cloud Computing in 2022 and the IEEE TCSVC Research Innovation Award from the IEEE CS Technical Community on Services Computing in 2023. He won the IEEE Region 1 Technological Innovation Award (Academic) in 2023. He was a recipient of the 2022-2023 International Science and Technology Cooperation Award and the 2023 Xiaoxiang Friendship Award of Hunan Province, China. He is a Member of the SUNY Distinguished Academy. Since 2020, he has been among the world's top few most influential scientists in parallel and distributed computing regarding single-year impact (ranked #2) and career-long impact (ranked #4) based on a composite indicator of the Scopus citation database. He is listed in Scilit Top Cited Scholars from 2023 to 2024 and is among the top 0.02% out of over 20 million scholars worldwide based on top-cited publications. He is listed in ScholarGPS Highly Ranked Scholars from 2022 to 2024 and is among the top 0.002% out of over 30 million scholars worldwide based on a composite score of three ranking metrics for research productivity, impact, and quality in the recent five years. He is an AAAS Fellow, an AAIA Fellow, an ACIS Fellow, and an AIIA Fellow. He is a member of the European Academy of Sciences and Arts. He is a member of Academia Europaea (Academician of the Academy of Europe)