

HyperJet: Joint Communication and Computation Scheduling for Hypergraph Tasks in Distributed Edge Computing

Kang Huang[†], Chao Qiu[†], Chenxuan Hou[†], Xiuhua Li[‡], and Xiaofei Wang[†]

[†]College of Intelligence and Computing, Tianjin University, Tianjin, China

[‡]School of Big Data & Software Engineering, Chongqing University, Chongqing, China

Email: {huangkang, chaoqiu, chenxuanhou}@tju.edu.cn, lixiuhua1988@gmail.com, xiaofeiwang@tju.edu.cn

Abstract—Distributed Edge Computing (DEC) has emerged as a novel paradigm, owing to its superior performance in communication latency, parallel computing efficiency, and energy consumption. With the surge of tasks in generative artificial intelligence, DEC faces higher demands for parallel computing efficiency. Scheduling multiple tasks for simultaneous processing, rather than one-by-one handling, could enhance parallel efficiency. *Multiple tasks have multi-dependencies, i.e., sequence dependency, attribute similarity, and attribute correlation.* Utilizing the bidirectional edges of traditional graphs to represent multi-dependencies can lead to an explosion in quantity. A hypergraph, with its hyperedges capable of connecting any number of vertices, can significantly solve the above problem. However, the multi-dependencies are rarely studied in the current research, posing the challenges, including *incapable representing and unable capturing of multi-dependency hypergraph*. In this work, we introduce a Joint communication and computation scheduling for hypergraph Tasks in DEC, namely *HyperJet*. To effectively represent multi-dependencies, we employ hypergraph construction to represent task attributes and utilize hypergraph partitioning to clarify and refine task attribute correlations, enhancing parallel efficiency. In response to the challenge of capturing multi-dependencies, we employ a scheduling mechanism with the hypergraph neural network that efficiently acquires higher-order attribute correlated information among convolution matrices, providing enriched contextual information on multi-dependencies that supports decision-making in scheduling tasks. The evaluations using real-world traces demonstrate an 18.07% improvement in parallel efficiency of task scheduling.

I. INTRODUCTION

Nowadays, by integrating edge computing and parallel computing, Distributed Edge Computing (DEC) is beginning to receive tremendous interest, which leverages the vast and abundant communication and computation resources at edges [1]–[4]. Particularly, DEC enhances parallel processing across user devices (UDs), reducing computation time [5] and overall energy consumption [6]. For instance, in autonomous driving applications, tasks such as vehicle motion analysis and image feature extraction can be distributed to nearby idle resource pools (RPs) to expedite the computation process while conserving energy consumption [7].

Since tasks in the application exhibit significant parallelism and *sequence dependency*, most studies utilize directed acyclic graphs (DAGs) to represent the applications, as shown in Fig. 1(a). They optimize the joint problem, i.e., latency and energy, by communication and computation scheduling [8]. Zhou *et*

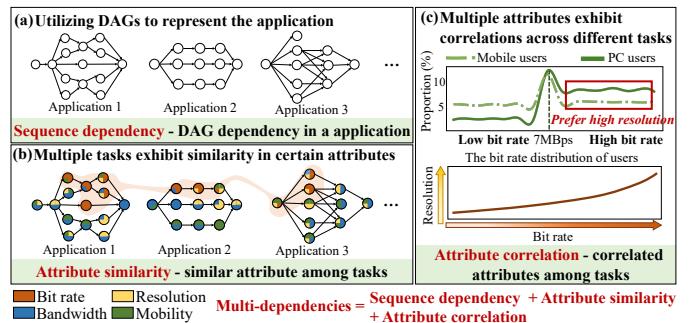


Fig. 1. Observations under different DAG applications.

al. in [9] consider a DAG cooperation approach to only share partial intermediate data. Tariq *et al.* in [10] only optimize the wasted idle of DAGs.

With the rise of generative artificial intelligence applications, there has been a significant increase in computational tasks [11], demanding higher parallel computing efficiency for DEC. Allocating only one or a few DAGs for computation significantly reduces parallel efficiency. We are investigating whether correlations exist among multiple DAGs that allow for simultaneous processing, rather than one-by-one handling, to enhance parallel efficiency. Thus, we investigate the request task data of live-streaming [12] and summarize the following two observations and issues.

Observation 1: Multiple tasks exhibit similarity in certain attributes: As shown in Fig. 1(b), application (i.e., DAG) 1, 2, 3 consist of several tasks, represented by dots. Each task has different attributes, such as bit rate, resolution, bandwidth, and mobility, represented by different colors. These attributes exhibit similarity among different DAGs, which is summarized as *attribute similarity – similar attributes among tasks*. One case is shown red shaded area of Fig. 1(b), i.e., these tasks all require a similar bit rate. **Issue 1:** In traditional graphs, applications sharing similar attributes may be linked by bidirectional edges. However, as the number of applications and attributes grows, the complexity escalates considerably.

Observation 2: Multiple attributes exhibit correlations across different tasks: As shown in Fig. 1(c), users with higher resolution attributes are more likely to use high bit rates, due to similar mobility patterns. This is likely because PC users prefer high-resolution live video streaming, leading

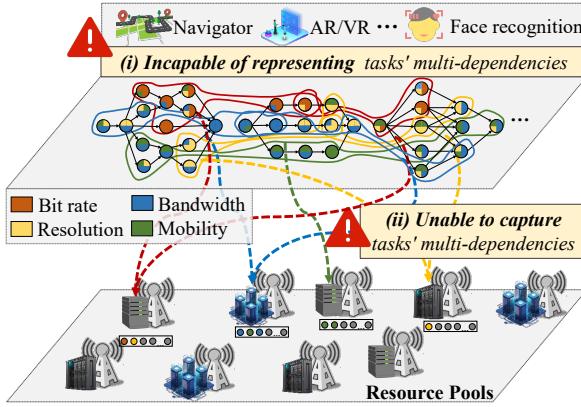


Fig. 2. The current challenges of parallel scheduling in DEC.

to higher bit rates. Thus, it can be observed that the attributes of resolution, bit rates, and mobility are correlated. It can be summarized as *attribute correlation – correlated attributes among tasks*. **Issue 2:** In traditional graphs, it is challenging to represent the correlations among three even more attributes using bidirectional edges.

Therefore, we have found that there are multi-dependencies among multiple DAGs, which can be referred to as *sequence dependency*, *attribute similarity*, and *attribute correlation*, as shown in Fig. 1. Traditional DAG dependencies can achieve a low level of parallelism for serial tasks while scheduling multiple DAGs with multi-dependencies can enhance parallel computational efficiency. However, representing multi-dependencies with bidirectional edges between two tasks leads to a combinatorial explosion in the number of attributes and edges, which significantly increases the complexity and contradicts the objective of enhancing parallelism.

A hypergraph, characterized by hyperedges that can connect any number of vertices, effectively addresses the issues above. Leveraging the advantages of hypergraphs, multiple DAGs could be uniformly represented as one or several hypergraphs. These hypergraph tasks are then scheduled for computation, significantly improving parallel efficiency [13]–[15].

However, the multi-dependencies are not studied in the current research, which poses challenges to efficient parallel scheduling in DEC, as shown in Fig. 2:

(i) Incapable of representing tasks' multi-dependencies.

The absence of modeling representations of tasks' multi-dependencies in scheduling leads to suboptimal decisions. The multi-dependencies among tasks are intricate, involving sequence dependency, attribute similarity, and attribute correlations. Most traditional models only represent sequence dependency as DAGs [16], focusing on order and overlooking other critical dependencies.

(ii) Unable to capture tasks' multi-dependencies.

Considering the complexity of dependencies between tasks, most studies have explored simple algorithms to solve scheduling problems. However, the methods involved are not fully adapted to extract information from intricate graph structures [17]. Therefore, there is a lack of effective methods to capture the impact of tasks' multi-dependencies on scheduling strategies.

In this work, we introduce a Joint communication and computation scheduling for hypErGraph Task in DEC, namely *HyperJet*. *HyperJet* leverages hypergraph-based modeling to simultaneously handle multiple DAGs, enhancing parallel efficiency, which is like the high-speed flight of a jet. The main contributions are summarized as:

- **Hypergraph-driven tasks' multi-dependency representation.** Our approach employs hypergraph construction to describe essential network attributes that extend beyond traditional DAG dependencies, such as bandwidth, mobility, and bit rate, providing richer and more comprehensive task information. Furthermore, we utilize hypergraph partitioning to explore the dependencies among multiple tasks characterized by different attributes. This integrated modeling enables the refinement of attribute correlations to facilitate parallel scheduling. Please refer to Section III.
- **High-order attribute correlations capturing in multi-dependencies.** This method utilizes the hypergraph neural network (HGNN) to fully exploit the comprehensive representational capabilities of hypergraphs, thereby capturing complex and high-order attribute correlations among convolutional matrices, providing enriched contextual information that supports decision-making in scheduling tasks. Please refer to Section IV.

Finally, we conduct evaluations of *HyperJet* using two real-world datasets for the task settings, including A and B Service Provider Traces. Our results demonstrate significant performance improvements, decreasing latency and energy consumption by 23.0% and 15.71%, respectively. Additionally, system utilization increases by 49.1%, resulting in an 18.07% improvement in parallel efficiency compared to GNN-based algorithms. The source code for the simulation environment with hypergraph information is available at this link: <https://github.com/xmcobwkr/HyperJet>.

II. SYSTEM MODEL AND PROBLEM FORMULATION

We aim to optimize the scheduling of multi-dependency tasks by utilizing effective learning methods to jointly communicate and compute, thereby minimizing the latency and energy consumption of DEC.

A. System Description

We propose a task scheduling scenario in DEC including a set of RPs, $\mathcal{N} = \{1, \dots, N\}$, a set of UDs, $\mathcal{M} = \{1, \dots, M\}$ of indexed by m , and a set of possible task requests $\mathcal{L}_m = \{1, \dots, L_m\}$ generated by UD m . All task requests in DEC are represented by $\mathcal{L} = \bigcup_{m \in \mathcal{M}} \mathcal{L}_m = \{1, \dots, i, \dots, L\}$. The i -th task can be denoted as a tuple $\{D_i, R_i\}$, where D_i and R_i represent the task size and execution result size. Each task can either be computed locally, or scheduled to RPs by base stations or access points. Thus, computing nodes consist of both RPs and UDs themselves and $\mathcal{F} = \{f_n \mid n \in \mathcal{N}\} \cup \{f_m \mid m \in \mathcal{M}\}$ denotes the central processing unit (CPU) processing frequency of computing nodes, where f_n and f_m denote the CPU frequency of the n -th RP and m -th UD. The

indicators $\mathcal{Z} \triangleq \{z_i \in \{0\} \cup \mathcal{N}\} \mid i \in \mathcal{L}\}$ denote task execution locations, where $z_i = 0$ represents local executed and $z_i \neq 0$ signifies scheduling to the RPs. Here, we assume UDs are unable to execute tasks generated by others.

B. Communication Model

In forward communication, each UD is assigned one subchannel of communication links for transmitting a scheduled task to the target RP with the transmission rates $r_{n,m}^{up}$, which can be expressed as

$$r_{n,m}^{up} = B_n^{up} \log_2 \left(1 + \frac{p_{n,m}^{up} |h_{n,m}^{up}|^2}{\delta_n^2} \right), \quad (1)$$

where B_n^{up} represents the wireless bandwidth, and δ_n^2 denotes the variance of complex white Gaussian channel noise. Furthermore, $p_{n,m}^{up}$ and $h_{n,m}^{up}$ denote the forward transmission power and the channel gain, respectively. Similarly, the transmission rate of the backward communication for results return can be calculated as $r_{n,m}^{down} = B_n^{down} \log_2 \left(1 + \frac{p_{n,m}^{down} |h_{n,m}^{down}|^2}{\delta_n^2} \right)$ with the same corresponding backward parameters definition. We consider that all task requests and results are allocated to distinct subchannels, which avoids co-channel interference.

C. Task Latency Model

1) *Individual Task Latency*: The latency associated with an individual task encompasses both transmission and computation latency. If the task is executed locally, the transmission delay can be eliminated. The transmission delay of i -th task, $T_i^{tr} = T_i^{up} + T_i^{down}$, consists of data upload time T_i^{up} and result download time T_i^{down} , which can be defined as

$$T_i^{up} = \mathbb{I}_{\{z_i \neq 0\}} D_i / r_{n,m}^{up}, \quad (2)$$

$$T_i^{down} = \mathbb{I}_{\{z_i \neq 0\}} R_i / r_{n,m}^{down}. \quad (3)$$

The computation latency is established by the ratio of task size and the computational capacity, which can be defined as

$$T_i^{co} = D_i / (\mathbb{I}_{\{z_i=0\}} f_m + \mathbb{I}_{\{z_i \neq 0\}} f_n). \quad (4)$$

However, due to the task dependency, which relies on the successful completion of all preceding tasks, the joint scheduling latency of i -th task cannot be simply obtained by adding its transmission and computation times.

2) *Dependent Task Latency*: In order to guarantee the accurate completion of individual tasks, we adhere to the principle that tasks should commence only after their corresponding parent tasks have been finished [18]. Consequently, the start time of a given scheduled task is contingent upon the maximum completion time of its parent tasks, thereby resulting in a dependency. Therefore, the completion time T_i^F of i -th task can be defined as

$$T_i^F = \max \left\{ T_i^A, \max_{j \in \text{AL}(i)} \{T_j^F\} \right\} + T_i^{co} + \mathbb{I}_{\{z_i \neq 0\}} T_i^{tr}, \quad (5)$$

where $\text{AL}(i)$ denotes parent task sets already executed before i -th task, and T_i^A denotes the available time of UD and RPs during scheduling, which can be defined as

$$T_i^A = \max \left\{ \mathbb{I}_{\{z_i=z_k\}} \{T_k^F\} \mid k = 1, \dots, i-1 \right\}, \quad (6)$$

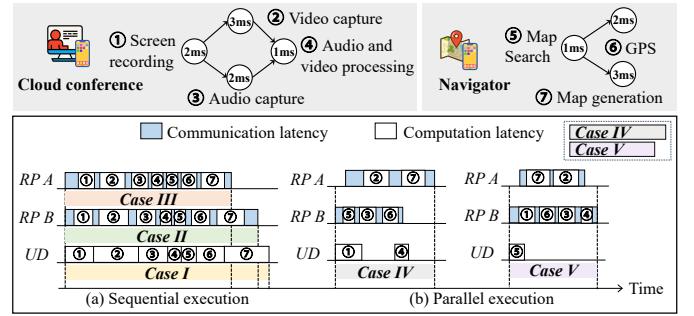


Fig. 3. Briefly example of task scheduling: (a) Sequential execution; (b) Parallel execution.

where indicator $\mathbb{I}_{\{z_i=z_k\}}$ represents the i -th task and the k -th task both scheduled on the same UD or RP.

After scheduling all tasks, we will obtain the set of task execution locations, $\mathcal{C}_{1:L} \triangleq \{c_i = z_i \in \{0\} \cup \mathcal{N}\}$. The total latency associated with this scheduling sequence is denoted as

$$T_{C_{1:L}} = \max_{i \in \mathcal{L}_w} \{T_i^F\}, \quad (7)$$

where $\mathcal{L}_w \subseteq \mathcal{L}$ represents the set of tasks without subsequent tasks. A brief explanation of the working principle of the task scheduling queue can be seen in Fig. 3.

D. Task Energy Consumption Model

An energy consumption model aims to evaluate the energy efficiency of task scheduling strategies, which is typically divided into the **computation energy consumption** and the **transmission energy consumption**.

1) *Computation energy consumption*: Energy consumed during task execution depends on the CPU frequency, computation latency, and power properties of UD and RPs, which can be defined as

$$E_i^{co} = \rho_q \times (f_{n,m}(q))^{\alpha_q} \times T_i^{co}, \quad (8)$$

where $f_{n,m}(q) : \mathcal{Q} \rightarrow \mathcal{F}$ is a mapping function that maps each q value in \mathcal{Q} to a unique CPU frequency of the computing nodes \mathcal{F} . The dynamic power exponent α_q (typically around 3) is generally presumed to be greater than 2, and ρ_q represents the power coefficient [19].

2) *Transmission energy consumption*: It refers to the energy consumed during the data transmission, including upload and download energy consumption. The amount of energy required to transfer i -th task can be formulated as

$$E_i^{up} = T_i^{up} \times p_{n,m}^{up}, \quad (9)$$

where $p_{n,m}^{up}$ represents the forward transmission power, which indicates the amount of energy consumed for data transmission per unit of time across various bandwidth conditions.

Similarly, the energy consumed during the downlink process is as follows:

$$E_i^{down} = T_i^{down} \times p_{n,m}^{down}, \quad (10)$$

where $p_{n,m}^{down}$ represents the backward transmission power. The transmission energy consumption can be calculated as

$$E_i^{tr} = E_i^{up} + E_i^{down}. \quad (11)$$

Therefore, $E_i = E_i^{co} + \mathbb{I}_{\{z_i \neq 0\}} E_i^{tr}$ denotes the total energy consumption of the i -th task, where $\mathbb{I}_{\{z_i \neq 0\}}$ indicates the transmission energy is considered only when execution locations belong to the set of RPs.

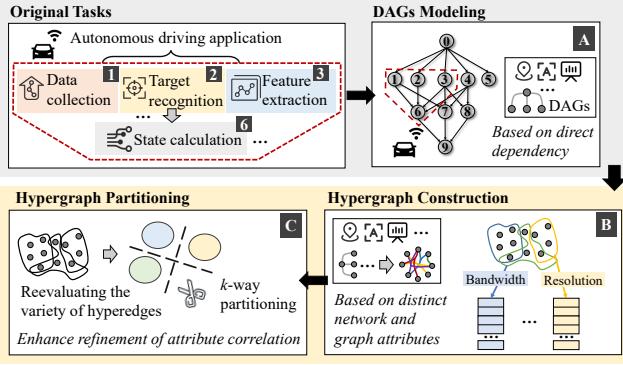


Fig. 4. The relationship between three subsections within Section III.

E. Problem Formulation

The objective of this study is to minimize the latency and energy consumption associated with task scheduling. Specifically, the performance metric is determined by calculating the weighted sum of latency and energy consumption [20]. Therefore, the optimization problem can be formulated as

$$(\mathbf{P}) \min_{C_{1:L}} \gamma_l \max_{x \in \mathcal{L}_w} \{T_x^F\} + \gamma_e \sum_{y \in \mathcal{L}} E_y, \quad (12)$$

where the coefficients γ_l and γ_e represent the weights for latency and energy consumption, respectively, ensuring dimensional consistency across the parameters. $C_{1:L}$ represents the set of task execution locations.

III. HYPERGRAPH-DRIVEN TASKS' MULTI-DEPENDENCY REPRESENTATION

Motivated by the goal to minimize latency and energy consumption as defined in Eq. (12), we introduce a hypergraph-driven approach for tasks' multi-dependency representation. This method provides additional task information for subsequent learning algorithm in Section IV, thereby preventing suboptimal optimization solutions. The detailed process is introduced in Fig. 4.

A. Dependent Computation Tasks

Typically, a particular application (e.g., autonomous driving) comprises multiple tasks (e.g., data collection and target recognition), each of which relies on the successful completion of all preceding tasks. We first model the applications as DAGs, denoted by $\mathcal{G} = \{\mathcal{L}, \mathcal{E}\}$, where the nodes of DAGs represent tasks and the set of edges $\mathcal{E} = \{\vec{e}_w \mid w = 1, \dots, E\}$ denotes dependencies pairs between tasks. Each directed edge \vec{e}_w is defined as $\vec{e}_w = (j, i)$, where the parent task j should be executed before the child task i , and each task can not be interrupted during execution.

B. Hypergraph Construction

To effectively represent the tasks' multi-dependencies, we develop a hypergraph construction model to provide a richer and more comprehensive view of tasks. We opted for a hypergraph as a representation method primarily because employing bidirectional edges in DAGs can result in an explosive increase in the number of edges, as shown in Fig. 5 derived from real-world traces utilized in our experiments.

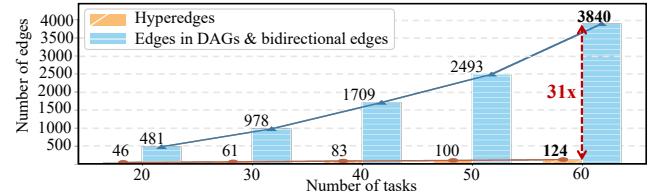


Fig. 5. The number of edges when representing multi-dependencies using bidirectional edges and hyperedges.

Definition 1. A Hypergraph \mathcal{H} denotes the dependency pairs $\mathcal{H} \triangleq (\mathcal{V}, \mathcal{E}_{hy})$, where \mathcal{V} is a non-empty finite vertex set and $\mathcal{E}_{hy} \triangleq \{e_{hy}^1, e_{hy}^2, \dots, e_{hy}^\phi\}$ is a family of subsets for \mathcal{V} named hyperedges such that $e_{hy}^\phi \neq \emptyset$ and $\cup_{\phi=1}^\Phi e_{hy}^\phi = \mathcal{V}$ [21].

For our considered scenario, a task corresponds to a vertex of \mathcal{H} , and the set of tasks is equal to the set of vertices of \mathcal{H} , i.e., $\mathcal{V} = \mathcal{L}$. We retain the original DAG's dependency in hypergraph representation. Consequently, the set of edges \mathcal{E} can be directly transformed into a set of 2-uniform hyperedges, maintaining the dependency from the DAG, denoted as

$$\mathcal{E}_{DAG} = \{\{v_j, v_i\} \mid (v_j, v_i) \in \mathcal{E}\}. \quad (13)$$

\mathcal{E}_{DAG} can fully cover the sequential dependency, which is the basic information required for task scheduling.

In order to accurately represent the dependencies between network attributes, we establish corresponding hyperedges that include network characteristics. For instance, a subset of vertices that have similar bandwidth attribute β_b can be represented as $N_a(\beta_b)$, and the corresponding hyperedges of attribute β_b can be formulated as

$$\mathcal{E}_a^{bw} = \{N_a(\beta_b) \mid \beta_b \in \mathcal{B}_b\}, \quad (14)$$

where \mathcal{B}_b is a collection of bandwidth attributes and models the dependencies in the attribute space at the group level. Similarly, other network attributes are also defined, following the same hyperedge principle. Specifically, $\mathcal{E}_a^{rs} = \{N_a(\beta_r) \mid \beta_r \in \mathcal{B}_r\}$, $\mathcal{E}_a^{br} = \{N_a(\beta_n) \mid \beta_n \in \mathcal{B}_{br}\}$ and $\mathcal{E}_a^{mo} = \{N_a(\beta_m) \mid \beta_m \in \mathcal{B}_m\}$ represent the different network attribute hyperedges, including resolution, bit rate, and mobility, respectively, which collectively constitute a network hyperedge set $\mathcal{E}_a^{Nw} = \{\mathcal{E}_a^{bw}, \mathcal{E}_a^{rs}, \mathcal{E}_a^{br}, \mathcal{E}_a^{mo}\}$.

To investigate the dependency of local connectivity on tasks, we constructed k -hop neighborhoods, $N_{H_k}(v) = \{b \mid \mathbf{A}_{bv}^k \neq 0, b \in \mathcal{V}\}$, defined as a collection of nodes separated by a distance of k -hop from one task node to another. The hyperedges of k -hop is represented as

$$\mathcal{E}_{Dist_k} = \{N_{H_k}(v) \mid v \in \mathcal{V}\}, \quad (15)$$

where \mathbf{A}_{bv}^k indicates the adjacency matrix and \mathcal{E}_{Dist_k} is possible to utilize the externally related vertices of the center task by expanding the search range in the graph structure. Therefore, the set of all hyperedges can be rewritten as $\mathcal{E}_{hy} = \{\mathcal{E}_{DAG}, \mathcal{E}_{Dist_k}, \mathcal{E}_a^{Nw}\}$ as shown in Fig. 6 (a).

C. Hypergraph Partitioning

Hypergraph construction realizes the multi-dependency representation from a task attributes perspective. By employing hypergraph partitioning, we explore the correlations among multiple tasks characterized by different attributes.

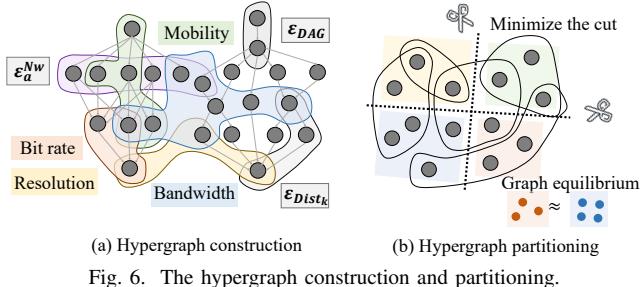


Fig. 6. The hypergraph construction and partitioning.

Definition 2. A k -way partition $\Gamma(\mathcal{H})$ of hypergraph \mathcal{H} involves dividing the vertex set of \mathcal{H} into k disjoint blocks $\Gamma(\mathcal{H}) = \{\mathcal{V}_1, \dots, \mathcal{V}_k\}$ [22]. These blocks have the properties that each vertex of \mathcal{H} belongs to exactly one block, and the union of all blocks equals the entire vertex set of \mathcal{H} .

In the selection of k -way partitioning strategies, our primary objective is to minimize the number of cutting edges. This approach preserves essential hyperedge attributes within the hypergraph and the cut metric $\text{cuts}(\cdot)$ can be written as

$$\text{cuts}(\Gamma(\mathcal{H})) = \sum_{i=1}^{|\mathcal{E}_{hy}|} (\lambda_i(\Gamma(\mathcal{H})) - 1), \quad (16)$$

where $\lambda_i(\Gamma(\mathcal{H})) \leq k$ denotes the number of partitions crossed by the i -th hyperedge. $\mathcal{E}_{hy_i} \in \mathcal{E}_{hy}$ in partition $\Gamma(\mathcal{H})$, named as hyperedge connectivity. A net, a subset of vertices related through edges, has its connectivity set defined for each block represented as $\Lambda_i(\Gamma(\mathcal{H})) = \{\mathcal{V}_j \in \Gamma(\mathcal{H}) \mid \mathcal{V}_j \cap \mathcal{E}_{hy_i} \neq \emptyset\}$. The hyperedge connectivity is defined as $\lambda_i(\Gamma(\mathcal{H})) := |\Lambda_i(\Gamma(\mathcal{H}))|$. We employ the balanced hypergraph partitioning algorithm KaHyPar [23] to minimize the cut:

$$\begin{aligned} & \min \quad \text{cuts}(\Gamma(\mathcal{H})), \\ & \text{s.t. } |\mathcal{V}_j| \leq (1 + \epsilon) \lceil \frac{|\mathcal{V}|}{k} \rceil, \mathcal{V}_j \in \Gamma(\mathcal{H}), \end{aligned} \quad (17)$$

where ϵ is the factor to balance the size of each partition. Re-evaluating the partitioning of various hyperedge attributes can form new hyperedges $\mathcal{E}_{pt} = \Gamma(\mathcal{H})$ to represent global information, as shown in Fig. 6 (b).

IV. HIGH-ORDER ATTRIBUTE CORRELATIONS CAPTURING FOR HYPERGRAPH TASK SCHEDULING

To exploit the comprehensive representational capabilities of hypergraphs, we introduce the design of HGNN to provide enriched contextual information that supports decision-making in scheduling tasks. Then, the dynamic optimization process is formulated into an Markov Decision Process (MDP).

A. HGNN Implementation for Task Representation Learning

HGNN updates task-related hidden states by aggregating essential features from hyperedges, which are linked to task attributes. When applying convolution operations on task hypergraphs, the system extracts further insights into task connectivity patterns and shared attributes from neighborhoods connected with hyperedges. Therefore, we apply hypergraph convolutional networks to each hypergraph to learn robust task representations that utilize higher-order attribute correlations.

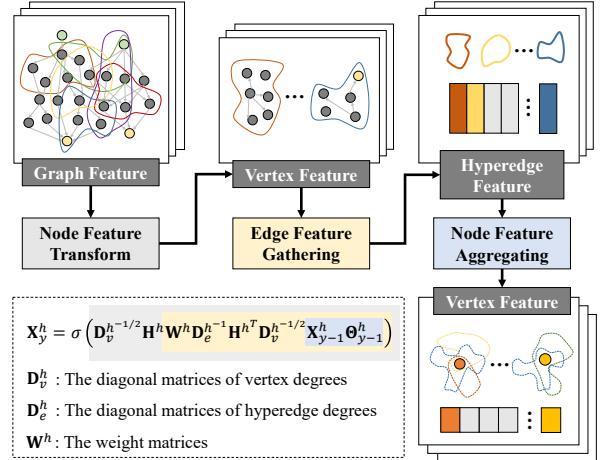


Fig. 7. The process of HGNN extracting hypergraph features.

The initial task graph h embedding matrix denotes as $\mathbf{X}_0^h = [\mathbf{x}_0^{h,l_1}, \dots, \mathbf{x}_0^{h,l_{|\mathcal{V}|}}]$ and the embedding of node \mathbf{x}_0^{h,l_1} encompasses the information for task scheduling, including the potential latency and energy consumption of transmission and computation process. Then, the updated tasks' representations \mathbf{X}_1 can be expressed as

$$\mathbf{X}_1^h = g(\mathbf{X}_0^h, \mathbf{H}^h, \mathbf{W}^h, \Theta_0^h), \quad (18)$$

where $g(\cdot)$ represents the hypergraph convolution process and \mathbf{H}^h denotes the $|\mathcal{V}| \times (|\mathcal{E}_{hy}| + |\mathcal{E}_{pt}|)$ incidence matrix of task hypergraph. The weight matrix \mathbf{W}^h is initialized as an identity matrix, which implies that all hyperedges are assigned equal weights. The transform matrix Θ_0 is used to extract the user features for propagation in the convolution layer.

For all tasks, the updating process can be derived as

$$\mathbf{X}_1^h = \mathbf{H}^h \mathbf{W}^h \mathbf{H}^{hT} \mathbf{X}_0^h \Theta_0^h, \quad (19)$$

where \mathbf{H}^{hT} represents the transpose of matrix \mathbf{H}^h . The matrix scale will be adjusted based on Eq. (19) when stacking y layers, potentially causing the gradient to disappear or explode during the feature propagation process. Thus, we implement symmetric normalization to preserve the numerical stability of the convolutional layer, similar to [24], and introduce a nonlinear activation function σ prior to output. Therefore, the hypergraph convolution layer can be computed as

$$\mathbf{X}_1^h = \sigma \left(\mathbf{D}_v^{h-1/2} \mathbf{H}^h \mathbf{W}^h \mathbf{D}_e^{h-1} \mathbf{H}^{hT} \mathbf{D}_v^{h-1/2} \mathbf{X}_0^h \Theta_0^h \right), \quad (20)$$

where \mathbf{D}_e^h and \mathbf{D}_v^h represent diagonal matrices of hyperedge and vertex degrees, respectively. For the task representation \mathbf{X}_1^h with the y -th layer, the output can be formulated by

$$\mathbf{X}_y^h = g(\mathbf{X}_{y-1}^h, \mathbf{H}^h, \mathbf{W}^h, \Theta_{y-1}^h), \quad (21)$$

$$= \sigma \left(\mathbf{D}_v^{h-1/2} \mathbf{H}^h \mathbf{W}^h \mathbf{D}_e^{h-1} \mathbf{H}^{hT} \mathbf{D}_v^{h-1/2} \mathbf{X}_{y-1}^h \Theta_{y-1}^h \right).$$

There are three steps through the calculation in Eq. (21): vertex feature transformation, hyperedge feature collection, and vertex feature aggregation shown in Fig. 7. By learning the interaction between nodes and hyperedges, HGNN can capture more detailed node features and edge relationships, which provides enriched contextual information that supports decision-making in scheduling tasks.

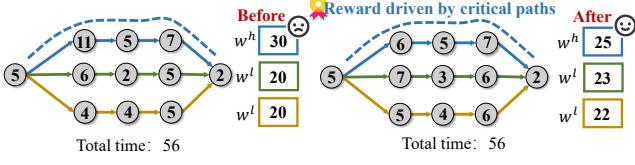


Fig. 8. The reward settings are based on critical paths.

B. Sequence Scheduling Problem Optimization

Initially, tasks include only the relevant scheduling details, such as transmission latency, energy consumption, and start time. From task representation learning, HGNN aggregates information in Eq. (21) from the task and other tasks connected through hyperedges to obtain a multi-dependency task embedding. We integrate hypergraph information into the task embeddings, allowing the MDP state to directly reflect task multi-dependencies, including topology, network properties, and attribute correlations, as shown in Fig. 9.

- **State S .** When the scheduling task i , the execution utility depends on its attributes and resource status. According to Eq. (5) and Eq. (11), the state is related to the scheduled task decisions before i . At time slot t , the state s_i includes the encoded hypergraph representation and the scheduling plan, i.e $s_i^t = \{D_i, R_i, \mathbf{X}_y^h, C_{1:i}^t\}$, composed of task size, received execution result size, task graph embeddings, and scheduling decision. Accordingly, the system state is defined as $\mathcal{S}^t = \{s_i^t \mid i = 1, 2, \dots, L\}$.

- **Action A .** Action space consists of discrete scheduling decisions, indicating the corresponding task execution position. The action space can be defined as $\mathcal{A}^t = \{a_i^t = c_i^t \mid i = 1, 2, \dots, L\}$, where 0 represents the local processing and others for corresponding RPs.

- **Reward driven by critical paths $\mathcal{R}(\mathcal{A}, \mathcal{S})$.** As Eq. (12) previously mentioned, the optimization objectives of our scheduling model are to reduce the average energy consumption and the task completion time. Specifically, unlike traditional rewards [25]–[27], our reward setting is inspired by **the critical path** [28], which is the longest path in a network diagram and determines the shortest possible duration for project completion in Fig. 8. Initially, the execution time for all tasks is calculated under identical computing resources, which does not affect the convergence. The path with the longest duration among these is identified as the critical path with the mark $\sigma_m = 1$ and others $\sigma_m = 0$. Then, during the scheduling process, the rewards are expressed as $R^t = -\Delta(\mathbb{I}_{\{\sigma_m==1\}}w^h(T_{C_{1:i}}^t - T_{C_{1:i-1}}^t) + \mathbb{I}_{\{\sigma_m==0\}}w^l(T_{C_{1:i}}^t - T_{C_{1:i-1}}^t)) - \Delta(E_i^t - E_{i-1}^t)$, where tasks on the critical path are assigned higher completion time weight coefficient w^h , whereas tasks on other paths are assigned lower weight coefficient w^l .

1) **Sequence-level Scheduling Mechanism:** Scheduling sequences vary in length due to the differing task counts within the hypergraph. The Sequence-to-Sequence (Seq2Seq) model is well-suited to handle these discrepancies, as it processes sequences of any length and generates outputs accordingly.

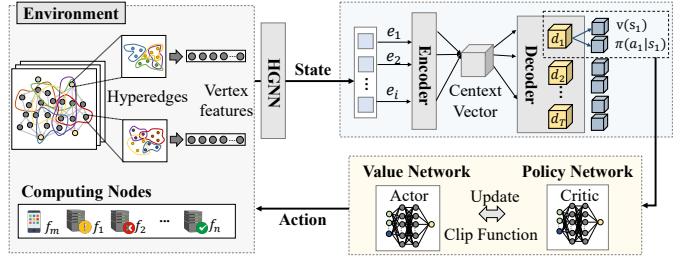


Fig. 9. Architecture of sequence scheduling problem optimization.

Consequently, we propose a **sequence-level scheduling mechanism** inspired by the Seq2Seq, with the Gated Recurrent Unit (GRU) network [29] for both the encoder and decoder.

Formally, the encoder's input is the task state sequence $\{s_i \mid i = 1, 2, \dots, L\}$, and the decoder's output is the scheduling actions $\{a_i \mid i = 1, 2, \dots, L\}$. At each step of encoding, the output e_i of the i -th encoder is derived as $e_i = g_e(s_i, e_{i-1})$, where $g_e(\cdot)$ denotes the function of the encoder. During each decoding step, the output of the decoder d_j is can be formulated by $d_j = g_d(d_{j-1}, a_{j-1}, v_j)$, where $g_d(\cdot)$ denotes the decoding functions and vector context v_j is represented by $v_j = \sum_{i=0}^L \mu_{ji} e_i$. μ_{ji} is the weight of the hidden layer state e_i in the encoder, which can be calculated with the softmax function given by

$$\mu_{ji} = \frac{\exp(\text{Dist}(d_{j-1}, e_i))}{\sum_{k=1}^L \exp(\text{Dist}(d_{j-1}, e_k))}, \quad (22)$$

where the function $\text{Dist}(\cdot)$ can assess the performance of the input at position i and the output at position j . Action a_j is derived by sampling from the policy $\pi(a_j \mid s_j)$ during training.

2) **Network and Policy Updates:** The network's objective is to identify the policy maximizing cumulative reward by

$$R = \sum_{i=1}^L R(s_i, a_i). \quad (23)$$

We employ the Proximal Policy Optimization (PPO) to train the network, aiming to achieve the overarching optimization objective function, derived as

$$L_{PPO}(\theta) = \mathbb{E}[L_C(\theta) - \tau L_V(\theta)], \quad (24)$$

where $L_C(\theta)$ and $L_V(\theta)$ denote the policy network and value network loss function, respectively, and τ represents the balance between two networks. Parameters θ determine a sequence of scheduling decisions.

Moreover, the policy network objective can be expressed as $L_C(\theta) = \mathbb{E}[\min(g_t(\theta) \hat{A}_t, g_{clip}(g_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t)]$, (25)

where the function $g_{clip}(\cdot)$ is defined to constrain $g_t(\cdot)$ within the interval $[1 - \epsilon, 1 + \epsilon]$ and ϵ represents the hyperparameter of clip range. \hat{A}_t is a Generalized Advantage Estimator (GAE) function at time slot t , which is expressed as $\hat{A}_t = \sum_{k=0}^{L-t+1} (\gamma \zeta)^k (R_{k+t} + \gamma V_\pi(s_{k+t+1}) - V_\pi(s_{k+t}))$, where ζ and $V_\pi(\cdot)$ denote the bias-variance trade-off and value function. The policy probability ratio $g_t(\theta)$ is given by

$$g_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta'}(a_t \mid s_t)}. \quad (26)$$

The objective function in PPO is updated by initially generating samples using the old policy $\pi_{\theta'}$, and subsequently

Algorithm 1 Training and Scheduling Process of *HyperJet*

```

1: Input:  $\mathbf{X}_0^h, \mathbf{H}^h$ 
2: for each episode do
3:   Initialize  $a_i^t = -1, i = 1, \dots, L$ 
4:   for iterations  $i \in \{1, \dots, L\}$  do
5:     for iterations  $k \in \{1, \dots, y\}$  do
6:        $\mathbf{X}_k^h \leftarrow g(\mathbf{X}_{k-1}^h, \mathbf{H}^h, \mathbf{W}^h, \Theta_{k-1}^h)$ 
7:     end for
8:     Sample action  $a_i$  with policy  $\pi(a_i^t | s_i^t)$ 
9:     Update  $T_i^F, T_i^A, E_i, \mathbf{X}_0^h$  with  $a_i^t$ 
10:    Calculate  $R^t$  and  $\hat{A}_t$  with  $T_i^F$  and  $E_i$ 
11:    Predict state value  $V_\pi(s_i^t)$  with critic network
12:    Update policy network parameter  $\theta' \leftarrow \theta + \alpha' \nabla_\theta L(\theta)$ 
13:   end for
14: end for

```

updating the current policy π_θ over training cycles. Therefore, the value network's squared error loss between the predicted state value $V_\pi(s_t)$ and the target state value $V_{\hat{\pi}}(s_t)$ is

$$L_V(\theta) = \mathbb{E}[V_\pi(s_t) - V_{\hat{\pi}}(s_t)]^2, \quad (27)$$

where $V_{\hat{\pi}}(s_t) = \sum_{k=0}^{L-t+1} \gamma^k R_{t+k}$ is estimated using the actual sampled data. Policy network parameter optimization updates can be expressed as

$$\theta' \leftarrow \theta + \alpha' \nabla_\theta L_{PPO}(\theta), \quad (28)$$

where α' denotes the learning rate via Adam optimizer [30].

Algorithm 1 presents the training process of *HyperJet*, which integrates hypergraph information into task embeddings using HGNN and then implements a task sequence scheduling.

Complexity analysis: The initial stage involves extracting the scheduling embedding information via the HGNN, with the time complexity of $O(yL(|\mathcal{E}_{hy}| + |\mathcal{E}_{pt}| + h_{emb} + h_{hggn}))$, where h_{emb} and h_{hggn} denotes embedding dimensions and HGNN hidden layer size. Subsequently, the policy network samples the scheduling action with the time complexity of $O(Lh_{s2s}(h_{emb} + h_{s2s}))$, where h_{s2s} denotes the hidden layer size of Seq2Seq network. As shown in Fig. 5 and Table I, the maximum number of hyperedges (i.e., 124) is less than the number of hidden units (i.e., 256), suggesting that the time complexity dimension of the two stages is relatively similar.

V. EXPERIMENTAL EVALUATION

We evaluate the joint scheduling capability of algorithms for tasks under different hypergraph representations and construct a simulation environment based on real-world datasets to replicate actual scheduling scenarios.

A. Experimental Environment

Task requests and resource pools. The data from the Alibaba Cluster Trace [31] comprises around 2.9 million tasks structured as DAGs, with parent node identifiers embedded within the task names and each DAG contains between 10 and 120 tasks. Since there are fewer DAGs with multiple nodes, we focus on DAGs with 10 to 30 tasks. The data from the PPIO Trace [32] contains 100,000 live stream request records, with essential parameters like data size, complexity, and attributes

TABLE I
THE HYPERPARAMETERS OF THE ALGORITHM SETUP

Hyperparameter	Value	Hyperparameter	Value
Batch Size	16	HGNN Layers	2
Encoder Layer Type	GRU	Decoder Layer Type	GRU
HGNN Hidden Units	128	Seq2Seq Hidden Units	256
Learning Rate (LR)	0.001	LR Decay Factor	0.98
Optimizer	Adam	Activation Function	ReLU
Discount Factor	0.95	GAE Lambda	0.95
PPO Clip Epsilon	0.2	Value Function Coefficient	0.25

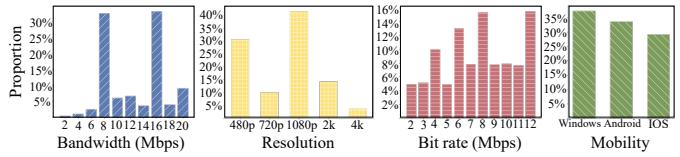


Fig. 10. Data distribution for constructing hypergraphs.

used to construct hypergraphs. The distribution of attributes is shown in the Fig. 10, which are used in the hyperedge set \mathcal{E}_a^{Nw} . We establish RPs consisting of 3 to 8 edge servers. Each RP, n , is configured with CPU frequency ranging from 1.0 to 3.5 GHz, 16 GB of memory, and 1 TB of hard disk capacity.

Algorithm setting. We implement the proposed Algorithm 1 based on PyTorch 1.13.1 with Python 3.9. The detailed settings are shown in Table I. To reduce oscillations during the training process, we add a scheduler named *ReduceLROnPlateau* to control the learning rate dynamically.

Evaluation metrics. In our research, we employ evaluation metrics including **reward**, **makespan**, **flowtime**, and **energy consumption** to assess parallel task structures. **Makespan** measures the longest completion time across tasks, reflecting scheduling efficiency [33]. **Flowtime** aggregates total task durations, evaluating overall system efficiency [34]. To enhance clarity, we transform the reward from negative values, which initially penalize higher latency and energy consumption, into positive values using the exponential function $\exp(x)$.

B. Tasks' Multi-dependencies Representation Comparison

Hypergraph construction efficiency: To verify the effectiveness of our proposed hypergraph construction method in representing tasks' multi-dependencies, we compare the performance with four baselines in Fig. 11: *i) DAG* [16] (tasks' the original DAG's dependency \mathcal{E}_{DAG}); *ii) DAG & k-hop* (tasks' original DAG's dependency \mathcal{E}_{DAG} and local connectivity structure \mathcal{E}_{Dist_k}); *iii) DAG & network* (tasks' original DAG's dependency \mathcal{E}_{DAG} and network attributes \mathcal{E}_a^{Nw}); *iv) DAG & k-hop & network* (tasks' original DAG's dependency \mathcal{E}_{DAG} , local connectivity structure \mathcal{E}_{Dist_k} and network attributes \mathcal{E}_a^{Nw}). As shown in Fig. 11 (a), the rewards for both types of hyperedges (\mathcal{E}_a^{Nw} & \mathcal{E}_{Dist_k}) are enhanced compared to the task hypergraph that only includes original DAG dependencies, demonstrating the efficacy of our proposed attribute hyperedges. However, the performance of three hyperedge types (0.493) in Fig. 11 (b) is nearly equivalent to the performance of two hyperedge types (0.496) in Fig. 11 (a). This indicates that simply adding attribute hyperedges does not necessarily improve scheduling efficiency.

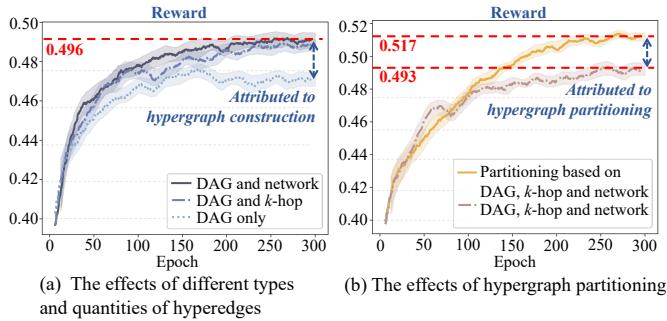


Fig. 11. Effects of hypergraph construction and partitioning.

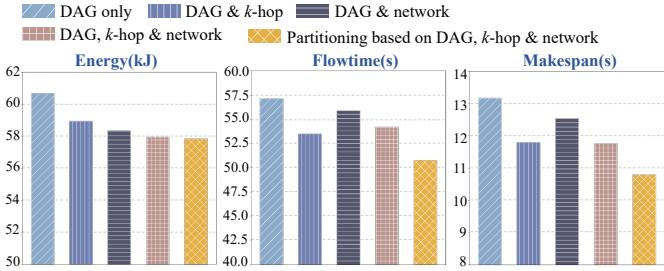


Fig. 12. System efficiency with hypergraph representation.

Hypergraph partitioning efficiency: Fig. 11 (b) illustrates the implementation of the hypergraph partitioning method, resulting in a 5% improvement in overall rewards. Hypergraph partitioning can integrate global information on the basis of complex task topologies, providing more critical representational information, including detailed insights into hyperedge internal structure and correlations between different hyperedges. Fig. 12 illustrates the enhancements in system latency and energy consumption achieved through hypergraph partitioning. Our method outperforms traditional DAG structures in terms of energy consumption, flowtime, and makespan, with reductions of 4.8%, 11.4%, and 23.0%, respectively.

C. Tasks' Multi-dependencies Capturing Comparison

Comparison baselines. We compared *HyperJet* with other scheduling methods in simulation environments with different number of tasks: (i) **DTODRL** [17]: This method uses a pre-trained Graph Neural Network (GNN) to capture the information of traditional DAGs; (ii) **GT-SAC** [20]: This approach employs the soft actor-critic algorithm to minimize the weighted sum of delay and energy consumption; (iii) **E-HEFT** [35]: This enhanced Heterogeneous Earliest Finish Time heuristic algorithm schedules tasks to the earliest available RP; (iv) **Greedy**: Tasks are scheduled to the RP with the best performance while considering load balancing; (v) **Random**: Tasks are randomly scheduled to RPs; (vi) **Local**: All tasks are executed locally; (vii) **Remote**: All tasks are scheduled to random remote RPs.

Algorithm performance. In Table II, *HyperJet* exhibits better latency and energy reduction compared to seven baseline models across different number of tasks. Specifically, it decreases the average latency by 28.76%, 41.06%, 42.39%, 32.79%, and 35.61%, respectively, while also achieving a makespan reduction of 1.37 seconds compared to GTODRL.

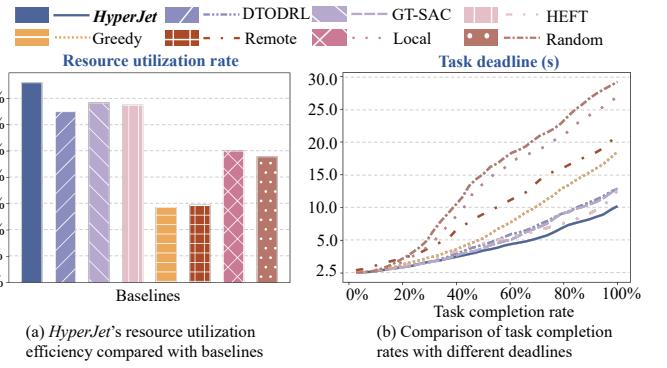


Fig. 13. System efficiency comparison of *HyperJet* and other baselines.

Additionally, *HyperJet* outperforms baselines in energy efficiency, showing reductions of 10.49%, 22.08%, 18.12%, 12.81%, and 15.12%. This dual advantage highlights *HyperJet*'s capability to handle increased tasks' dependencies more effectively than GTODRL and GT-SAC. By leveraging task attribute hyperedges, *HyperJet* provides an innovative approach that more effectively represents and captures high-order data correlations compared to traditional GNNs. While E-HEFT focuses on completion time by selecting more powerful but higher energy-consuming RPs, *HyperJet* optimizes the trade-off between latency and energy consumption.

Parallel efficiency. In Fig. 13, we compare *HyperJet* to other baselines in resource utilization and task completion rates. *HyperJet* achieves an impressive average resource utilization increase of 49.1%, showcasing its superiority in parallel scheduling. Additionally, Fig. 13 (b) maintains a high task completion rate across various deadline constraints. At 10 seconds, *HyperJet* almost reaches a 98% completion rate, while other baselines only achieve a maximum completion rate of 83%. This means that *HyperJet* achieves an 18.07% improvement in parallel efficiency compared to DTODRL. Notably, even E-HEFT, which emphasizes time efficiency, exhibits lower completion effectiveness than *HyperJet*. By allowing more tasks to be scheduled for computation simultaneously, *HyperJet* optimizes the overall utilization of resources and effectively reduces the task competition time.

D. Auxiliary Validation

Training efficiency. To further validate *HyperJet*, we perform tests through reward settings. By integrating critical paths into our reward configurations, the flowtime shows a reduction of 8.1% over 300 epochs and the makespan experiences a 14.2% decrease in Fig. 14. These analyses demonstrate the direct impact of reward settings driven by critical paths on system performance, confirming their crucial role in optimizing task scheduling and improving training efficiency in *HyperJet*.

Learning performances. Fig. 15 compares *HyperJet* performance to five baselines: (I) w/o HGNN: without HGNN for the network; (II) w/o Seq2Seq: without Seq2Seq for the network; (III) w/o hypergraph: without representation of tasks' multi-dependencies; (IV) w/o hypergraph & HGNN: without HGNN in the original tasks' structure; (V) w/o hypergraph & HGNN: without Seq2Seq in the original tasks'

TABLE II
COMPARISON OF OTHER SCHEDULING METHODS WITH DIFFERENT NUMBER OF TASKS

Number of tasks	Learning Algorithms				Heuristic Algorithms								
					Latency (s) — Energy (kJ)				Heuristic Algorithms				
	<i>HyperJet</i>	DTODRL [17]	GT-SAC [20]	E-HEFT [35]	Greedy		Random		Local		Remote		
20	8.09	35.72	8.32	36.32	8.60	36.21	8.92	42.48	10.77	39.38	11.16	42.87	19.19
30	9.15	41.00	10.53	43.92	11.00	43.23	12.48	55.13	15.92	57.20	16.99	60.23	22.62
40	10.33	56.15	13.16	60.23	13.15	59.99	12.62	71.97	18.63	72.97	17.23	75.65	29.22
50	14.76	74.94	15.91	76.27	15.82	77.41	14.52	90.09	20.69	85.83	21.47	93.08	40.56
60	17.82	95.45	19.10	98.45	20.50	100.8	19.89	117.4	28.81	118.1	25.85	123.2	47.88
													38.37
													12.57
													43.72

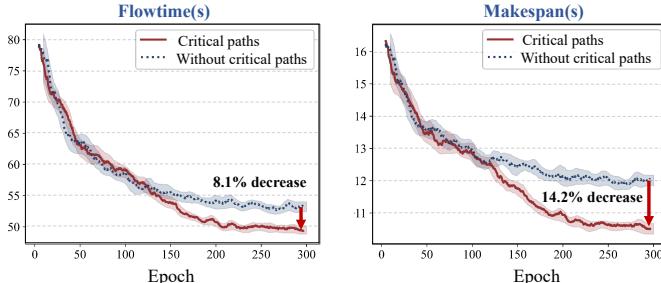


Fig. 14. The reward setting efficiency with the critical paths.

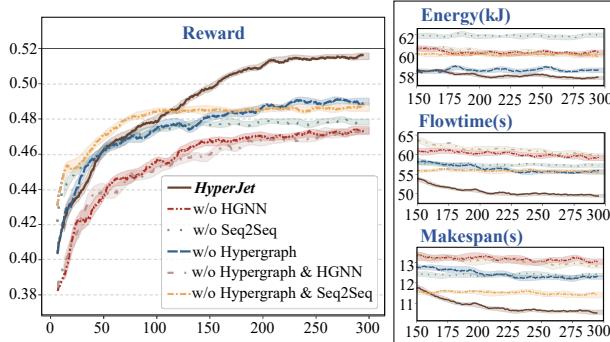


Fig. 15. Ablation study of *HyperJet* components performances.

structure. *HyperJet* outperforms baseline (II)-(V) in overall reward performance. Compared with baseline (III) demonstrates that incorporating hypergraph representations into task structures effectively models more detailed task dependencies, thereby addressing the challenge of representing tasks' multi-dependencies. Similarly, compared with baseline (I), *HyperJet* enhances the capture of tasks' multi-dependencies, providing effective support for task parallel scheduling.

VI. RELATED WORKS

Many studies are trying to enhance the parallel scheduling efficiency of DEC but frequently encounter issues associated with attribute similarity and correlations across different tasks.

Dependency task scheduling for DEC. Optimization of scheduling decisions for dependent tasks enhances parallel computing efficacy. Shu *et al.* [36] represent applications as DAGs to delineate fine-grained dependencies. A heuristic algorithm is a viable solution [37], [38] for optimizing task scheduling in DEC. Zhe *et al.* [39] introduce a collaborative evolution approach to address the dependency task scheduling problem. As the complexity of the task dependency topology escalates, the algorithm's search space expands, leading

heuristic algorithms to frequently encounter local optima. Consequently, Jin *et al.* [16] employ a learning approach with Seq2Seq network to attain optimal solutions for latency and energy consumption but lack attention to task state embedding. Although [17], [27] consider task state embedding, they either disregard the dependencies between attributes or exclusively consider the state representation of DAG original attributes.

Hypergraph learning for multi-dependencies. Recently, hypergraphs have proven highly effective in modeling complex dependencies. In work [40], they emphasize the significance of hypergraphs in system throughput optimization by characterizing more intricate resource mapping relationships. Zhang *et al.* [41] suggest a hypergraph matching algorithm based on local search to address the energy consumption issue of scheduling. In work [42], hypergraph modeling is applied to heterogeneous networks to achieve a more refined expression of resource dependencies. Lin *et al.* [43] combine the information represented by hypergraphs with reinforcement learning, but neglect to consider whether the training network could capture multi-dependencies.

VII. CONCLUSION

In this paper, we have introduced *HyperJet* to improve the parallel efficiency of task scheduling. Our approach has employed hypergraph construction and partitioning to represent and refine task attribute correlations and utilized HGNN to provide enriched contextual information on multi-dependencies to support task parallel scheduling. Comprehensive experiments have been conducted, demonstrating the effectiveness of our proposed *HyperJet*. However, *HyperJet* focuses on uncovering the relationships between attributes, which may lead to efficiency reductions in single attribute task scheduling scenarios. In the future, research may focus on enhancing its adaptability to dynamic attribute variations.

ACKNOWLEDGEMENT

This work was supported in part by the Tianjin Natural Science Foundation General Project under Grant No. 23JCY-BJC00780, the National Science Foundation of China under Grant No. 62072332, the Beijing-Tianjin-Hebei Collaborative Basic Research Project on Key Technologies for Efficient Environmental Sensing and Situation Simulation of Intelligent Connected Vehicles under Grant No. F2024201070, and the Tianjin Xinchuang Haihe Lab under Grant No. 22HHX-CJC00002. Thanks to PPIO Cloud Computing Co., Ltd for supporting this work.

REFERENCES

- [1] Q. Chen, K. Wang, S. Guo, T. Shi, J. Li, Z. Cai, and A. Zomaya, “Latency-optimal pyramid-based joint communication and computation scheduling for distributed edge computing,” in *IEEE INFOCOM 2023*, pp. 1–10, 2023.
- [2] Y. Yao, B. Liu, Y. Zhao, and W. Shi, “Towards edge-enabled distributed computing framework for heterogeneous android-based devices,” in *IEEE/ACM SEC 2022*, pp. 531–536, 2022.
- [3] X. Gong, “Delay-optimal distributed edge computing in wireless edge networks,” in *IEEE INFOCOM 2020*, pp. 2629–2638, 2020.
- [4] K. Bhardwaj, A. Gavrilovska, V. Kolesnikov, M. Saunders, H. Yoon, M. Bondre, M. Babu, and J. Walsh, “Addressing the fragmentation problem in distributed and decentralized edge computing: A vision,” in *IEEE IC2E 2019*, pp. 156–167, 2019.
- [5] S. Duan, D. Wang, J. Ren, F. Lyu, Y. Zhang, H. Wu, and X. Shen, “Distributed artificial intelligence empowered by end-edge-cloud computing: A survey,” *IEEE Commun. Surv. Tutorials*, vol. 25, no. 1, pp. 591–624, 2023.
- [6] Z. Shuai, Z. Hu, J. Gai, Y. Chen, J. Chen, H. Zhang, and F.-Y. Wang, “Metaverse-enabled intelligence for open-terrain field vehicle fleets: Leveraging parallel intelligence and edge computing,” *IEEE Trans. Intell. Veh.*, vol. 9, no. 2, pp. 3111–3116, 2024.
- [7] C. Chen, Y. Zhang, Z. Wang, S. Wan, and Q. Pei, “Distributed computation offloading method based on deep reinforcement learning in ICV,” *Appl. Soft Comput.*, vol. 103, p. 107108, 2021.
- [8] H. Chang, M. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee, “Scheduling in MapReduce-like systems for fast completion time,” in *IEEE INFOCOM 2011*, pp. 3074–3082, 2011.
- [9] X. Zhou, S. Ge, P. Liu, and T. Qiu, “DAG-based dependent tasks offloading in mec-enabled IoT with soft cooperation,” *IEEE Trans. Mob. Comput.*, vol. 23, no. 6, pp. 6908–6920, 2024.
- [10] U. U. Tariq, H. Ali, L. Liu, J. Hardy, M. Kazim, and W. Ahmed, “Energy-aware scheduling of streaming applications on edge-devices in IoT-based healthcare,” *IEEE Trans. Green Commun. Netw.*, vol. 5, no. 2, pp. 803–815, 2021.
- [11] M. Xu, H. Du, D. Niyato, J. Kang, Z. Xiong, S. Mao, Z. Han, A. Jamalipour, D. I. Kim, X. Shen, V. C. M. Leung, and H. V. Poor, “Unleashing the power of edge-cloud generative AI in mobile networks: A survey of aigc services,” *IEEE Commu. Sur. Tutorials*, vol. 26, no. 2, pp. 1127–1170, 2024.
- [12] S. Shen, Y. Feng, M. Xu, C. Zhang, X. Wang, W. Wang, and V. C. Leung, “A holistic QoS view of crowdsourced edge cloud platform,” in *IEEE/ACM IWQoS 2023*, pp. 01–10, 2023.
- [13] C. Li, Y. Zhang, Z. Hao, and Y. Luo, “An effective scheduling strategy based on hypergraph partition in geographically distributed datacenters,” *Comput. Networks*, vol. 170, apr 2020.
- [14] B. Cheng, X. Guan, and H. Wu, “A hypergraph based task scheduling strategy for massive parallel spatial data processing on master-slave platforms,” in *Geoinformatics 2015*, pp. 1–5, 2015.
- [15] H. Zhao, X. Liu, and X. Li, “Hypergraph-based task-bundle scheduling towards efficiency and fairness in heterogeneous distributed systems,” in *IEEE IPDPS 2010*, pp. 1–12, 2010.
- [16] J. Wang, J. Hu, G. Min, W. Zhan, A. Y. Zomaya, and N. Georgalas, “Dependent task offloading for edge computing based on deep reinforcement learning,” *IEEE Trans. Comput.*, vol. 71, no. 10, pp. 2449–2461, 2022.
- [17] Z. Cao, X. Deng, S. Yue, J. Ren, and J. Gui, “Dependent task offloading in edge computing using GNN and deep reinforcement learning,” *IEEE Internet Things J.*, vol. 11, no. 12, pp. 21632–21646, 2024.
- [18] 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, 2023.
- [19] B. Hu, Y. Shi, and Z. Cao, “Adaptive energy-minimized scheduling of real-time applications in vehicular edge computing,” *IEEE Trans. Ind. Informatics*, vol. 19, no. 5, pp. 6895–6906, 2023.
- [20] J. Li, B. Gu, Z. Qin, and Y. Han, “Graph tasks offloading and resource allocation in multi-access edge computing: A drl-and-optimization-aided approach,” *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 6, pp. 3707–3718, 2023.
- [21] M. N. Soorki, W. Saad, M. H. Manshaei, and H. Saidi, “Social community-aware content placement in wireless device-to-device communication networks,” *IEEE Trans. Mob. Comput.*, vol. 18, no. 8, pp. 1938–1950, 2019.
- [22] T. Heuer, “A direct k -way hypergraph partitioning algorithm for optimizing the steiner tree metric,” in *ALENEX 2024* (R. Chowdhury and S. P. Pissis, eds.), pp. 15–31, 2024.
- [23] L. Gottesbüren, M. Hamann, S. Schlag, and D. Wagner, “Advanced Flow-Based Multilevel Hypergraph Partitioning,” in *SEA 2020*, vol. 160 of *LIPICS*, pp. 11:1–11:15, 2020.
- [24] Y. Gao, Y. Feng, S. Ji, and R. Ji, “Hgnn⁺: General hypergraph neural networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 3, pp. 3181–3199, 2023.
- [25] M. Maray, E. Mustafa, J. Shuja, and M. Bilal, “Dependent task offloading with deadline-aware scheduling in mobile edge networks,” *Internet of Things*, vol. 23, p. 100868, 2023.
- [26] 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 Distributed Syst.*, vol. 32, no. 1, pp. 242–253, 2021.
- [27] Y. Li, J. Li, Z. Lv, H. Li, Y. Wang, and Z. Xu, “GASTO: A fast adaptive graph learning framework for edge computing empowered task offloading,” *IEEE Trans. Netw. Serv. Manag.*, vol. 20, no. 2, pp. 932–944, 2023.
- [28] Q. Liu, C. Wang, X. Li, and L. Gao, “An improved genetic algorithm with modified critical path-based searching for integrated process planning and scheduling problem considering automated guided vehicle transportation task,” *J. Manuf. Syst.*, vol. 70, pp. 127–136, 2023.
- [29] Q. Zeng, S. Guo, R. Cao, Z. Zhao, and H. Duan, “Legal transition sequence recognition of a bounded petri net using a gate recurrent unit,” *IEEE Trans. Big Data*, vol. 10, no. 1, pp. 66–76, 2024.
- [30] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *ICLR 2015*, 2015.
- [31] “Alibaba-clusterdata.” <https://github.com/alibaba/clusterdata>. [Online].
- [32] S. Shen, Y. Feng, M. Xu, C. Zhang, X. Wang, W. Wang, and V. C. Leung, “A holistic qos view of crowdsourced edge cloud platform,” in *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*, pp. 01–10, 2023.
- [33] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, “Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends,” *Swarm Evol. Comput.*, vol. 62, p. 100841, 2021.
- [34] V. Fernandez-Viagas, P. Perez-Gonzalez, and J. M. Framinan, “The distributed permutation flow shop to minimise the total flowtime,” *Computers & Industrial Engineering*, vol. 118, pp. 464–477, 2018.
- [35] J. Mack, S. E. Arda, Ü. Y. Ogras, and A. Akoglu, “Performant, multi-objective scheduling of highly interleaved task graphs on heterogeneous system on chip devices,” *IEEE Trans. Parallel Distributed Syst.*, vol. 33, no. 9, pp. 2148–2162, 2022.
- [36] C. Shu, Z. Zhao, Y. Han, G. Min, and H. Duan, “Multi-user offloading for edge computing networks: A dependency-aware and latency-optimal approach,” *IEEE Internet Things J.*, vol. 7, no. 3, pp. 1678–1689, 2020.
- [37] Y. Zhai, W. Sun, J. Wu, L. Zhu, J. Shen, X. Du, and M. Guizani, “An energy aware offloading scheme for interdependent applications in software-defined iov with fog computing architecture,” *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 6, pp. 3813–3823, 2021.
- [38] M. Zhao, J. Yu, W. Li, D. Liu, S. Yao, W. Feng, C. She, and T. Q. S. Quek, “Energy-aware task offloading and resource allocation for time-sensitive services in mobile edge computing systems,” *IEEE Trans. Veh. Technol.*, vol. 70, no. 10, pp. 10925–10940, 2021.
- [39] Q. Xiao, J. Zhong, L. Feng, L. Luo, and J. Lv, “A cooperative coevolution hyper-heuristic framework for workflow scheduling problem,” *IEEE Trans. Serv. Comput.*, vol. 15, no. 1, pp. 150–163, 2022.
- [40] M. N. Soorki, W. Saad, M. H. Manshaei, and H. Saidi, “Social community-aware content placement in wireless device-to-device communication networks,” *IEEE Trans. Mob. Comput.*, vol. 18, no. 8, pp. 1938–1950, 2019.
- [41] L. Zhang, H. Zhang, L. Yu, H. Xu, L. Song, and Z. Han, “Virtual resource allocation for mobile edge computing: A hypergraph matching approach,” in *IEEE GLOBECOM 2019*, pp. 1–6, 2019.
- [42] Q. Hao, M. Sheng, D. Zhou, and Y. Shi, “A multi-aspect expanded hypergraph enabled cross-domain resource management in satellite networks,” *IEEE Trans. Commun.*, vol. 70, no. 7, pp. 4687–4701, 2022.
- [43] K. Lin, H. Wang, B. Chen, and G. Fortino, “Hypergraph-based autonomous networks: Adaptive resource management and dynamic resource scheduling,” *IEEE Commun. Stand. Mag.*, vol. 6, no. 3, pp. 16–22, 2022.