Collaborative Inference for Deep Neural Networks in Edge Environments

Meizhao Liu[1], Yingcheng Gu[1], Sen Dong[2], Liu Wei[1], Kai Liu[1], Yuting Yan[2], Yu Song[1], Huanyu Cheng[1], Lei Tang[1], and Sheng Zhang[2*]

[1]State Grid Jiangsu Electric Power Co Ltd, Information & Telecommunication Branch, Nanjing, China

[2]State Key Lab. for Novel Software Technology, Nanjing University, Nanjing, 210023 China

[e-mail: sheng@nju.edu.cn]

*Corresponding author: Sheng Zhang

Received November 21, 2023; revised January 16, 2024; accepted June 12, 2024; published July 31, 2024

Abstract

Recent advances in deep neural networks (DNNs) have greatly improved the accuracy and universality of various intelligent applications, at the expense of increasing model size and computational demand. Since the resources of end devices are often too limited to deploy a complete DNN model, offloading DNN inference tasks to cloud servers is a common approach to meet this gap. However, due to the limited bandwidth of WAN and the long distance between end devices and cloud servers, this approach may lead to significant data transmission latency. Therefore, device-edge collaborative inference has emerged as a promising paradigm to accelerate the execution of DNN inference tasks where DNN models are partitioned to be sequentially executed in both end devices and edge servers. Nevertheless, collaborative inference in heterogeneous edge environments with multiple edge servers, end devices and DNN tasks has been overlooked in previous research. To fill this gap, we investigate the optimization problem of collaborative inference in a heterogeneous system and propose a scheme CIS, i.e., collaborative inference scheme, which jointly combines DNN partition, task offloading and scheduling to reduce the average weighted inference latency. CIS decomposes the problem into three parts to achieve the optimal average weighted inference latency. In addition, we build a prototype that implements CIS and conducts extensive experiments to demonstrate the scheme's effectiveness and efficiency. Experiments show that CIS reduces 29% to 71% on the average weighted inference latency compared to the other four existing schemes.

Keywords: Edge intelligence, Collaborative inference, Deep inference, Computation offloading.

This work was supported in part by the Science and Technology Project of State Grid Co., LTD (Research on data aggregation and dynamic interaction technology of enterprise-level real-time measurement data center, 5108-202218280A-2-399-XG).

http://doi.org/10.3837/tiis.2024.07.003 **ISSN : 1976-7277**

1. Introduction

In recent times, deep neural network (DNN), positioned as a cornerstone technology for Artificial Intelligence (AI) and Machine Learning (ML) [1], has achieved remarkable development. This technology has been widely applied in various fields including Computer Vision [2], Natural Language Processing [3] and speech recognition [4].

Nevertheless, with the improvement of universality and accuracy, the scale of DNN model is also growing, which means more memory and computational resources are required. For instance, when executing inference on a 224x224 image using VGG16, it entails processing over 138 million parameters through more than 15 billion operations. If executed on a Nexus 5 smartphone, the task would take approximately 16 seconds [5]. This is obviously intolerable for some real-time tasks. Consequently, to meet the memory and computation requirements, DNN inference tasks are typically offloaded to cloud servers with extensive computational resources. However, this traditional cloud computing paradigm encounters several challenges. First, it struggles to meet the real-time requirements of some Internet of Things (IoT) applications when the network condition is poor. Then, massive data may impose a considerable burden on network communication and cloud server processing. What's more, concerns regarding privacy leaks due to data transmission to the cloud also cannot be ignored [6]. To address these problems, device-edge collaborative inference has emerged as a promising paradigm to promote edge intelligence.

Model partition is an important technology in collaborative inference. Motivated by the significant reduction of data size of some intermediate layers compared to that of input layer, a DNN model is partitioned so that the inference task can be sequentially executed on the end device and edge server. Proper partition can make full use of the computational resources of servers within limited communication overhead [7]. Most prior research on collaborative inference has been limited in the single scenario involving single task and single server. However, realistic scenarios often encompass the presence of multiple edge servers (ESs) and multiple end devices (EDs) with distinct DNN tasks. Meanwhile, different end devices and edge servers, encompassing smartphones, base stations, and gateways, may exhibit different computational capacities, forming the heterogeneous edge environments, in which collaborative inference needs to be considered.

This paper studies the DNN partition, task offloading and scheduling problem in heterogeneous collaborative inference systems, which aims to minimize the average weighted inference latency for DNN tasks. In this problem, each task can be partitioned at different layers according to the computational capacities of devices and network conditions, so we must **determine the layers at which the DNN task is partitioned, i.e., partition strategy**. Prior explorations have predominantly limited to the DNNs with chain topology. However, many advanced DNN models adopt DAG topology, e.g., GoogleNet [8] and ResNet [9], which brings new challenges to collaborative inference. Besides, each task can be offloaded to one of the servers in the system, so we must **determine the server to which the task is offloaded, i.e., offloading strategy**. What's more, there can be more than one task offloaded to the same server, so we must **determine the order in which the tasks are executed, i.e., scheduling strategy**. The FCFS(First-Come-First-Serve) policy is commonly taken by previous works [11]. However, in real-world scenarios, different tasks often have different priorities. For example, in a smart home system, the priority of tasks responsible for security systems needs to be higher than those responsible for other tasks (such as audio control). When multiple tasks are offloaded to the same server, the scheduling strategy has an undeniable impact on their weighted inference latency. Hence, FCFS policy can hardly adapt to this priority scenario.

To fill these gaps, this paper deeply studies the collaboration of EDs and ESs in a heterogeneous scenario. We formulate this problem as a ILP problem and denote it as $POSP$, short for *task Partition, Offloading and Scheduling Problem*. Then a heuristic scheme CIS, i.e., collaborative inference scheme, is proposed for $POSP$. The main contributions of this paper are summarized as follows:

1) This paper puts forward the collaborative inference in a heterogeneous scenario. The stated problem seeks to minimize the average weighted inference latency by optimizing partition strategy, scheduling strategy and offloading strategy.
2) This paper builds a system model for heterogeneous collaborative inference, and proposes a scheme CIS to minimize average weighted inference latency based on this model. CIS decouples the optimization problem into three subproblems: DNN partition, task offloading and task scheduling.
3) Based on DADS [7], a widely used scheme for DNN partition, algorithm MCP is proposed to deal with the partition for different DNN models, no matter what the topology it is. Then we design SWRTF policy for the task scheduling problem since they have different priorities. At last, CIS utilizes branch and bound to obtain the final strategies, which traverse all feasible solutions in a breadth first manner with proper pruning.
4) Extensive experiments are conducted to verify the performance of this scheme. The comprehensive and in-depth analysis of the results demonstrates that our scheme can greatly reduce the inference latency compared with current approaches.

2. Related Work

Collaborative inference is a significant research direction in edge intelligence, which means end devices complete the DNN inference tasks with the assistance of edge servers or cloud servers.

Kang et al. [16] initially proposed layer-wise partition of DNN models as an approach to enable collaborative inference. However, their approach is limited to linearly-structured DNNs and proved ineffective for more general Directed Acyclic Graph (DAG) structured DNNs. Given that many DNNs exhibit DAG structures, Hu et al. [7] modeled the partition of these DNNs as a min-cut problem and provided a method for computing optimal partition points using max-flow solutions. On this basis, they introduced a system named DADS (Dynamic Adaptive DNN Splitting) that can handle model partition in dynamic network environments. Zhang et al. [17] noted that the min-cut-based partition method has a high time complexity, making it challenging to adapt to scenarios with rapidly changing network conditions. Consequently, they simplified the problem and introduced a two-stage system called QDMP for finding the optimal partitioning point. Wang et al. [18] proposed a hierarchical scheduling optimization strategy called DeepInference-L. By executing computations and data transfers between layers in a pipelined manner, they further reduced the overall latency of collaborative inference. Furthermore, Duan et al. [19] considered the scenarios where multiple DNN inference tasks run on a single mobile device. They employed convex optimization techniques to comprehensively address multi-task partition and scheduling strategies. However, these studies only consider the scenarios of a single device and a single server, which is not applicable to general edge computing scenarios.

Gao et al. [10] designed a dynamic evaluation strategy under a time slot model, dividing a DNN inference task into multiple subtasks and dynamically determining its offloading strategy. Tang et al. [11] proposed an iterative alternative optimization (IAO) algorithm to solve the

problem of task partition in a multi-user scenario. Mohammed et al. [12] proposed that in the context of fog computing, a DNN model can be divided into multiple parts, each of which can be executed at fog nodes or locally. Combined with matching theory, an adaptive dynamic task partition and scheduling system DINA was proposed, which can greatly reduce the inference latency. Although the aforementioned studies take the multiple devices into consideration, they ignore the fact that offloading all tasks to a single edge server would lead to issues of excessive load on that server and underutilization of resources on other servers.

To address this problem, Yang et al. [13] introduced an edge-device collaborative inference system called CoopAI, which employs a novel partition algorithm to offload a DNN inference task onto multiple edge servers. By analyzing the characteristics of DNN inference, it permits servers to pre-fetch necessary data, reducing the cost of data exchange and consequently reducing inference latency. Liao et al. [14] delved into the DNN partitioning and task offloading challenges in heterogeneous edge computing scenarios. They conducted an analysis of the task offloading issue involving multiple terminal devices and multiple edge servers. Employing an optimal matching algorithm, they proposed an algorithm that comprehensively addresses both partitioning and offloading concerns, thereby reducing overall system inference latency and energy consumption. Shi et al. [15] presented an offline partitioning and scheduling algorithm, GSPI, for enhancing the speed of DNN inference tasks in a multi-user multi-server setting. However, it's important to note that they exclusively considered scenarios where all users execute the same DNN inference task.

This paper focuses on the collaborative DNN inference problem under the scenario with multiple end devices and multiple edge servers. Given the varying computational capacities of end devices and their distinct upload bandwidths to different edge servers, the manner in which DNN models are partitioned and the selection of servers for tasks to offloading significantly impact the inference latency. By comprehensively considering multiple factors, we propose the collaborative inference scheme CIS for heterogeneous edge computing environments.

3. System Model and Problem Formulation

We first introduce the heterogeneous collaborative inference system mentioned above in Section **3.1**. Then we formalized our problem with the target of minimizing average weighted inference latency and the decision variable involving partition strategy P, offloading strategy X and scheduling strategy Φ in Section **3.2**.

3.1 Heterogeneous Collaborative Inference System

An edge computing system is comprised of a set of end devices and a set of resource-constrained edge servers. As shown in **Fig. 1**, each ED is equipped with a pretrained DNN model and executes the DNN inference task of this model. To accelerate the execution of DNN inference tasks, each DNN model can be partitioned at layer-level and then offloaded to one of the ESs. The EDs and ESs are connected in a LAN, where each ES is accessible for each ED.

We denote the set of n end devices as $D = \{d_1, d_2, \cdots, d_n\}$. For convenience, we use task j to denote the task on d_j. Each end device d_j is associated with two parameters: w_j and $Cap_d(j)$. Here w_j represents the priority of task j and task with greater w_j has a higher priority. $Cap_d(j)$ represents the computational capacity of ED d_j measured in FLOPS. It worth nothing that in the heterogeneous system, these parameters of different EDs can be varying. In this paper, we assume all the tasks can be partitioned at most once, which means each task can

only be offloaded to at most one server.

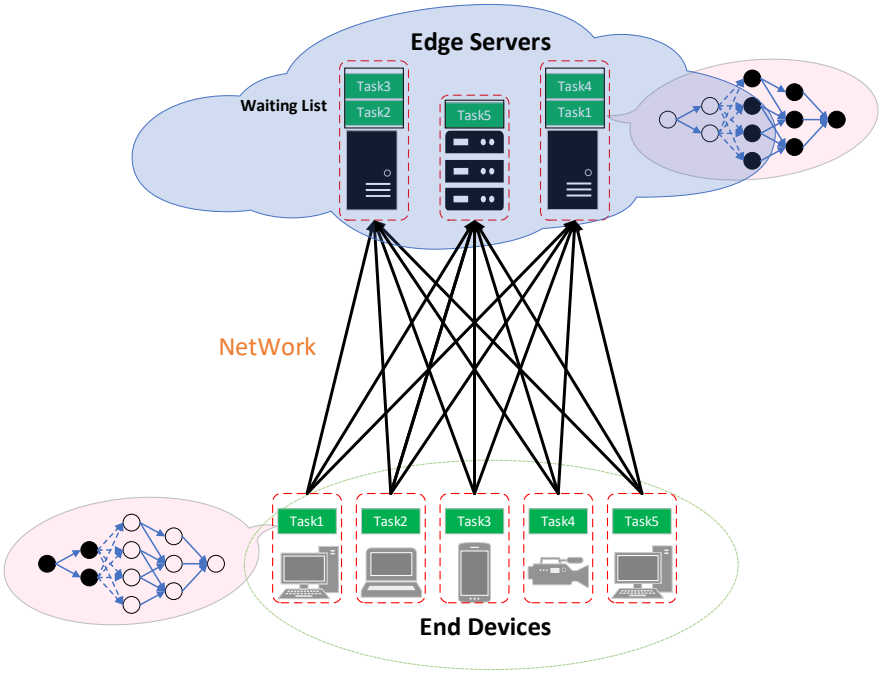


Fig. 1. Collaborative inference system in heterogeneous edge computing scenarios

We denote the set of m edge servers as $S = \{s_1, s_2, \cdots, s_m\}$. Let $Cap_s(i)$ denote the computational capacity of s_i, measured in FLOPS. Let b_{ij} denote the bandwidth between s_i and d_j. In this paper, we assume that each b_{ij} is given and constant. ESs will pre-load the DNN models of the tasks offloaded to them. After intermediate data being sent to the server, the task will be added to a waiting list to wait for scheduling. Once a server is idle, it will select a task from the waiting list to execute, and the execution can't be interrupted until the task is finished. **Table 1** lists the main symbols used in this article.

Table 1. Main notations

Symbol	Definition
s_i	The ith edge server
d_j	The jth end device
w_j	The priority of task j
$Cap_s(i)$	The computational capacity of ES s_i (FLOPS)
$Cap_d(j)$	The computational capacity of ED d_j (FLOPS)
b_{ij}	The bandwidth between s_i and d_j
$\Phi(s_i)$	The scheduling strategy of tasks offloaded to s_i
p_j	The partition strategy of task j
$L_j(p)$	The local computation size of task j with partition p (FLOPs)
$C_j(p)$	The transmission data size of task j with partition p (FLOPs)
$R_j(p)$	The remote computation size of task j with partition p (FLOPs)
x_{ij}	Binary decision variable, $x_{ij} = 1$ indicates that task j is offloaded to ES s_i
t_{ij}^{local}	The latency of local computing when task j is offloaded to ES s_i
t_{ij}^{trans}	The latency of data transmission when task j is offloaded to ES s_i

t_{ij}^{wait}	The latency of waiting for scheduling when task j is offloaded to ES s_i
t_{ij}^{remote}	The latency of remote computing when task j is offloaded to ES s_i
T	The average weighted inference latency of the system

3.2 System Model

3.2.1 DNN Layer-level Computation and Output data Model

DNN models are usually composed of a series of layers, such as convolutional layers, excitation layers, active layers, pooling layers and fully connected layers. To compute the inference latency of a DNN, we must analyze the computation and output data size of each layer of the DNN model.

Layer-level Computational Cost: We measure the computational cost of each layer using FLoating point OPerations (FLOPs), which represents the number of basic mathematical operations (such as addition, subtraction, multiplication, etc.) to be performed. Similar methods have been used in [14]. Let $com(v)$ denote the computational cost of layer v. Since some layers, like active layer, have a very small computational cost, we just consider the main DNN layers whose computational cost has impact on the inference latency of the model as follows:

- Convolutional Layer: Convolutional layer is one of the most basic layers in DNNs. It performs convolution operations on the input data through a set of convolution kernels to extract local features at different locations. The computational cost of convolution layer depends on the size of the input feature map and the size and number of the convolution kernel. For the convolution layer v, assuming the size of the input feature map is $w_{in} \times h_{in}$, the size of the convolution kernel is $w_k \times h_k$, and the number of channels of the input feature map is C_{in}, the number of channels of the output feature map is C_{out}, the size of stride is $w_s \times h_s$, then its computational cost is:
$$com(v) = \left(\frac{w_{in} - w_k}{w_s} + 1\right) * \left(\frac{h_{in} - h_k}{h_s} + 1\right) * C_{in} * C_{out} * w_k * h_k * 2, \quad (1)$$
where $\left(\frac{w_{in}-w_k}{w_s} + 1\right) * \left(\frac{h_{in}-h_k}{h_s} + 1\right)$ represents the number of multiplicative operations required for each output position and it is also the number of additive operations required for each output position.
- Fully Connected Layer: Fully connected layer is also one of the most basic layers in deep neural networks. By connecting each input neuron to an output neuron and giving each connection a weight, the features extracted from the previous layers are combined and integrated to generate the final output. For fully connected layer v, assuming that the dimension of the input feature vector is d_{in} and the dimension of the output feature vector is d_{out}, then its computational cost is:
$$com(v) = (d_{in} + (d_{in} - 1)) * d_{out}, \quad (2)$$
where d_{in} denotes the multiplicative operation, and $(d_{in} - 1)$ denotes the additive operation.

Output Data Size: We use $data(v)$ to denote the output data size of layer v. For layer v, assuming the size of its output feature map is $w_{out} \times h_{out}$, then the output data size of this layer v is:
$$data(v) = C_{out} * w_{out} * h_{out}, \quad (3)$$

If the tensor size of the input image is ($3 \times 224 \times 224$), the computation and output data size of the layers of MobileNet_V2 model are shown in **Fig. 2**.

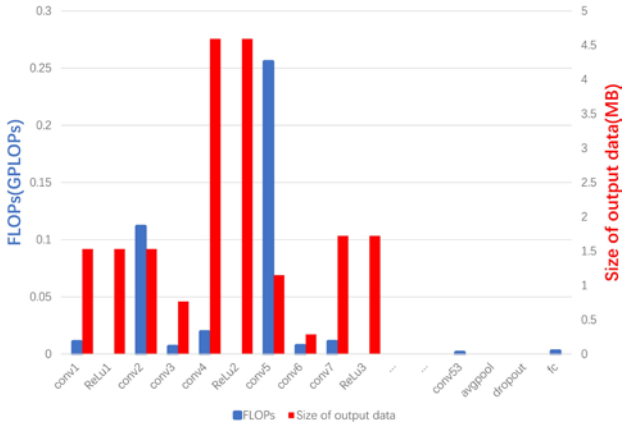
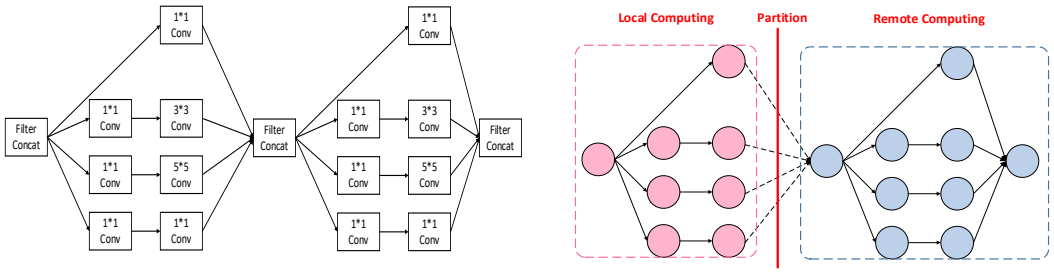


Fig. 2. Computation and output data size of each layer of MobileNetV2 model.

3.2.2 DNN Partition Model

The inference process of DNN is actually a process of forward propagation, starting from the input layer and gradually moving forward, each layer conducts a series of calculations on its own input and sends the results to its subsequent layers as their input. Thus, given a DNN model M, we can represent M as a DAG (directed acyclic graph) $G =< V, E >$, where $v_i \in V$ corresponds to one layer and the directed edge $e_{ij} \in E$ represents the dependency between v_i and v_j. It should be emphasized that each vertex may have multiple edges starting from it and multiple edges ending at it. For example, **Fig. 3(a)** shows a piece of GoogLeNet [8], which can be modeled as a DAG as shown in **Fig. 3(b)**.



(a) **(b)**

Fig. 3. A piece of GoogLeNet (a) and the DAG corresponding to it (b)

In the context of deep neural networks (DNNs), it should be noted that the computational cost and output size of each layer are different and independent of each other, which provides an opportunity for DNN partition. DNN partition is to divide a DNN model into two parts to execute them on different devices. Formally, we define $p_j =< V_j^l, V_j^r >$ as the partition of task j which partitions its vertex set V_j into two disjoint subsets V_j^l and V_j^r. The layers corresponding to the vertex in V_j^l are executed on an ED, and the layers corresponding to the vertex in V_j^r are executed on an ES. **Fig. 3(b)** shows a partition of GoogLeNet mentioned above.

Thus, as to task j and one of its partitions p_j, the local computational cost is:

$$L_j(p_j) = \sum_{v \in V_j^l} com(v), \tag{4}$$

the transmission data size is:

$$C_j(p_j) = \sum_{v \in V_j^c} data(v), \tag{5}$$

and the remote computational cost is:

$$R_j(p_j) = \sum_{v \in V_j^r} com(v), \tag{6}$$

where V_j^l and V_j^r denote the set of layers executed on the EDs and the set of layers executed on the ESs respectively. V_j^c represents the set of layers that need to send their output to the ES, which means each layer in V_j^c is belong to V_j^l and has a successor layer in V_j^r.

3.2.3 Task Scheduling Model

In reality, there are often fewer edge servers than end devices, so it is common for multiple tasks to be offloaded to the same server. In our system, the server can only execute one task at a time, so tasks need to wait for scheduling before execution. As a result, the scheduling policy, specifically the execution order of tasks, has a great impact on the average weighted inference latency. In previous related works, they schedule the tasks in a first-come-first-service (FCFS) manner [11], but it can't solve our problem well for tasks have different priorities. For example, suppose there are three tasks offloaded to the same server. We define the arrival time of a task as the time it takes before the intermediate data is sent to the server, including local computing time and data transmission time. Also, we the server computing time as the time of task execution on the server. Then the three task's arrival time, server computing time and task priority are (5,5,3), (7,2,2), (3,6,1) respectively. **Fig. 4** shows the results of two scheduling strategies, where the left one is according to FCFS and the right one is in another way. Their average weighted inference latencies are 83/3 and 24 respectively, which shows that different scheduling strategies have an important impact on inference latency. To formally represent the scheduling strategy, let $\phi(s_i)$ denote the task sequence on s_i, then the scheduling strategy of the system can be represented as $\Phi = \{\phi(s_1), \phi(s_2), \cdots, \phi(s_m)\}$. The k*th* scheduled task on s_i can be represented as $\phi_k(s_i)$, where $1 \le k \le |\phi(s_i)|$.

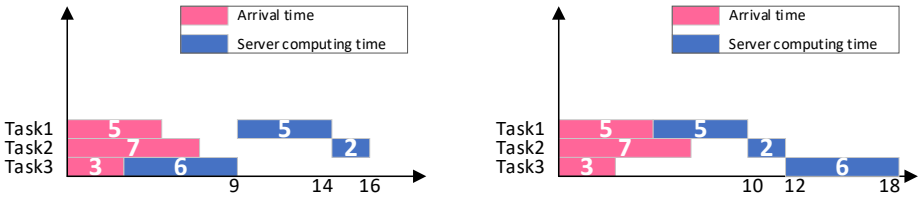


Fig. 4. Two different scheduling strategies

3.2.4 Task Offloading and Inference Latency Model

In the heterogeneous system, due to the disparity in computational capacity and bandwidth between ESs, it is obvious that the task offloading strategy does affect the inference latency. Formally, we use a set of binary variables $X = \{x_{11}, x_{12}, \cdots, x_{1m}, \cdots, x_{n1}, x_{n2}, \cdots, x_{nm}\}$ to denote the offloading strategy. Specifically, $x_{ij} = 1$ if and only if task j is offloaded to ES s_i; otherwise, $x_{ij} = 0$. Since one task can only be offloaded to one ES, we can get that $\sum_{i=1}^{m} x_{ij} = 1, \forall 1 \le j \le n$.

Once a task is partitioned and decided to be offloaded to an ES, the execution of this task can be divided into four stages: local computing, data transmission, waiting for scheduling and remote computing. Therefore, if task j is partitioned by p_j and offloaded to ES s_i, its inference latency is:

$$T_{ij} = t_{ij}^{local} + t_{ij}^{trans} + t_{ij}^{wait} + t_{ij}^{remote}, \tag{7}$$

where t_{ij}^{local} denotes the latency of local computing:

$$t_{ij}^{local} = L_j(p_j)/Cap_d(j), \tag{8}$$

t_{ij}^{trans} denotes the latency of data transmission:

$$t_{ij}^{trans} = \frac{C_j(p_j)}{b_{ij}}, \tag{9}$$

t_{ij}^{wait} denotes the latency of waiting for scheduling:

$$t_{ij}^{wait} = max\left(t_{ij'}^{wait} + t_{ij'}^{remote}, t_{ij}^{local} + t_{ij}^{trans}\right) - \left(t_{ij}^{local} + t_{ij}^{trans}\right), \tag{10}$$

where j' denotes the task scheduled before j on s_i. Suppose j is $\phi_k(s_i)$, then j' is $\phi_{k-1}(s_i)$. t_{ij}^{remote} denotes the latency of remote computing:

$$t_{ij}^{remote} = R_j(p_j)/Cap_s(i). \tag{11}$$

Assuming that all tasks start at the same time, the average weighted inference latency of the entire system is:

3.2.5 Problem Formulation

The QoE [23] of the applications based on DNN model improves with the reduction of inference latency. Since different tasks have different priorities, we try to minimize the average weighted inference latency T of the system by collaborative inference. Considering the computational capacity and network bandwidth of the system, an inference reduction problem is formulated in this subsection. We now refer to this optimization problem with partition strategy P, offloading strategy X and scheduling strategy Φ as *POSP*, e.g. task Partition, Offloading and Scheduling Problem, and define it as follows:

$$\begin{aligned} POSP: \quad & \min_{P,\Phi,X} T \\ s.t. \quad & C1{:}\ V_j^l \cup V_j^r = V_j, V_j^l \cap V_j^r = \emptyset, \forall 1 \le j \le n, \\ & C2{:}\ \sum_{i=1}^{m} x_{ij} \le 1, \forall\ 1 \le j \le n, \\ & C3{:}\ x_{ij} = 1\ or\ 0, \forall\ 1 \le j \le n, \forall\ 1 \le i \le m, \\ & C4{:}\ \phi(s_i) = \{j |\ x_{ij} = 1\}. \end{aligned} \tag{12}$$

The optimization objective function T is the average weighted inference latency of the entire system, which can be formulated as:

$$T = \frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij} w_j T_{ij}. \tag{13}$$

Constraint $C1$ guarantees the valid of partition strategy of each task. Constraint $C2$ and $C3$ manifest that one task can only be offloaded to one ES at most or even just be completed locally. Constraint $C4$ is used to guarantee t task sequence on s_i is consistent with offloading strategy X. Due to the existence of multiple variables and the high coupling between variables, problem *POSP* is too complex to be solved directly. Thus, we need to give further analysis and decompose problem *POSP* to get the optimal strategies.

4. Algorithm Design

We have modeled our problem as an optimization problem *POSP* and point out that it is complex owing to the existence of multiple decision variables. In this section, we first reveal several structural properties of our problem and then decompose the problem into three parts. After that, we propose an algorithm MCP for DNN partition, a scheduling policy SWRTF for task scheduling and an algorithm BBO for DNN allocation to solve *POSP* step by step.

4.1 Problem Decomposition

Since there are multiple sets of variables in *POSP* and the variables are coupled with each other, we need to decouple them to decompose the complex problem. The main idea of our scheme is to give the corresponding partition strategy P and scheduling strategy Φ for each offloading strategy X, thus binding P and Φ to X, which means as long as X is determined, P and Φ can be determined accordingly. Then we just need to solve the new problem $\mathcal{P}$ which only has X as its decision variables. So, our scheme can be decomposed into three steps: 1) give the partition strategy P for each offloading strategy X, 2) give the scheduling strategy Φ for each offloading strategy X, 3) generate the new problem $\mathcal{P}$ which only has X as its decision variables and solve this problem to give the optimal offloading strategy X. Since step 1) and 2) has bound P and Φ to X, so with the result of step 3), we can give the final optimal strategy P, Φ and X for problem *POSP*.

Fig. 5. CIS flow

4.1.1 DNN Partition

For most DNN, they have many different partitions, which makes it more difficult for us to solve the problem. However, some partitions will lead to excessive inference latency thus almost impossible to become the optimal solution. Therefore, we can reduce the solution space by selecting an optimal partition p_{ij} for each DNN task j when trying to offload it to ES s_i. Based on the method proposed in [7], we design an algorithm MCP, e.g., min-cut based partition, which first constructs a latency graph G' for each task j and ES s_i based on the DAG G of the DNN model of task j to convert the optimal partition problem to the minimum weighted s–t cut problem of G', and then get the optimal partition p_{ij} using a min-cut algorithm.

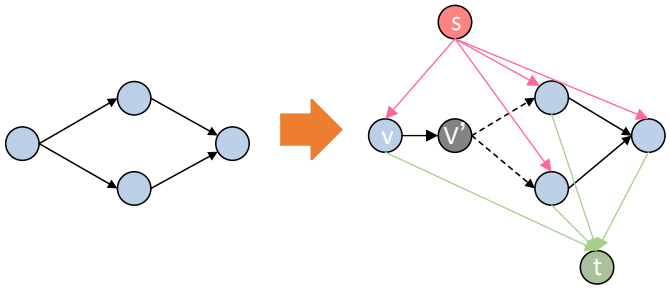


Fig. 6. Constructing the latency graph

For task j and ES s_i, let $G =< V, E >$ denote the corresponding DAG of the DNN model of task j, we can construct a weighted DAG G', e.g., its latency graph, as follows:

1) Add a source node s and a sink node t to G'.
2) Add the remote computing edges E_{remote}: For each node $v \in V$, add an edge from s to v, whose weight is $com(v)/Cap_s(i)$, e.g., the time for executing this layer on s_i.
3) Add the local computing edges E_{local}: For each node $v \in V$, add an edge from v to t, whose weight is $com(v)/Cap_d(j)$, e.g., the time for executing this layer on d_j.
4) Add the data transmission edges E_{trans}: It is worth nothing that the output data of a layer only need to be transmit at most once even if it has more than one successor layers. Thus, there are two conditions we add data transmission edges.
   a) For each node $v \in V$, if it only has one successor v' in G, add an edge from v to v', whose weight is $\frac{C_j(data(v))}{b_{ij}}$, i.e., the time for transmitting v's output data from d_j to s_i.
   b) For each node $v \in V$, if it has more than one successors in G, we first add a virtual node $v_{virtual}$ to G', then add an edge from v to $v_{virtual}$, whose weight is $\frac{C_j(data(v))}{b_{ij}}$, and then add an edge from $v_{virtual}$ to all the successors of v in G, whose weight is positive infinity.

Fig. 6 shows an example of constructing the latency graph of a DAG. At this stage, the optimal partition problem has been converted to the minimum weighted s–t cut problem of the latency graph G'. For a s-t cut C of G', the edges in C comprise three parts: remote computing edges, local computing edges and data transmission edges. Thus, the value of C is exactly equal to the inference latency of the DNN task at the partition of C without considering the time waiting for scheduling. Then we use a min-cut algorithm to get the minimum weighted s–t cut of G', and get the optimal partition p_{ij} from this. In the following steps of determining scheduling and offloading strategies, we suppose task j is partitioned at p_{ij} when offloaded to s_i. Now, we have bound P to X, i.e., the partition strategy P can be determined accordingly when offloading strategy X is determined. Since the time complexity of min-cut algorithm is $O(|V|^2|E|\ln|V|)$, where $|V|$ is the number of nodes in the DAG and $|E|$ is the number of edges in the DAG, the time complexity of constructing the latency graph is $O(|V|+|E|)$, the time complexity of MCP is $O(mn|V'|^2|E'|\ln|V'|)$, where $|V'|$ is the largest number of nodes in all the DAG and $|E'|$ is the largest number of edges in all the DAG.

Algorithm 1. MCP
Input: The computational capacity $Cap_d(j)$ of ED d_j and the DAG G_j of the DNN model on d_j, the computational capacity $Cap_s(s)$ of ES s_i, the bandwidth b_{ij} between d_j and s_i.
1: for each ED $d_j \in \boldsymbol{D}$ do
2: for each ES $s_i \in \boldsymbol{S}$ do
3: $G' \leftarrow$ latency_graph_construct(G_j, $Cap_d(j)$, $Cap_s(s)$, b_{ij});
4: $p_{ij} \leftarrow$ min_cut(G');
5: return $P = \{p_{ij} | 1 \le j \le n, 1 \le i \le m\}$;

4.1.2 Task Scheduling

To give the scheduling strategy Φ for each offloading strategy X, we design a new scheduling policy for tasks offloaded to the same ES called "shortest weighted remaining time first" (SWRTF). In particular, SWRTF has three rules:

1) The ES will not be idle unless there are no tasks in the waiting list.
2) Tasks are executed non-preemptively, that is, once a task starts executing, other tasks must wait until the execution of the task ends.

3) The task with shortest weighted remaining time RT_j will be scheduled first, where $RT_j = t_{ij}^{remote}/w_j$.

For a given task j and ES s_i, if task j is decided to be offloaded to s_i, then the partition p_{ij} of the task j are given as Section **4.1.1** mentioned. Suppose the set of tasks offloaded to s_i is $J_i = \{j_1, j_2, \cdots, j_q\}$. Then for each j_k in J_i, the local computation $L_{j_k}(p_{ij_k})$, transmission data size $C_{j_k}(p_{ij_k})$ and remote computation $R_{j_k}(p_{ij_k})$ of j_k are given. Thus $t_{ij_k}^{local}$, $t_{ij_k}^{trans}$ and $t_{ij_k}^{remote}$ can be obtained from Eqs. (8)(9)(11). Thus, the arrival time, i.e., the time of local computing and data transmission, and the shortest weighted remaining time of each task are given. Then the scheduling strategy $\phi(s_i)$ for these tasks can be given based on SWRTF. The same goes for other ESs. Now, we have bound Φ to X, i.e., scheduling strategy Φ can be determined accordingly when offloading strategy X is determined.

4.1.3 Task Offloading

Based on the practical consideration on DNN partition and task scheduling in Section **4.1.1** and **4.1.2**, we can bind partition strategy P and scheduling strategy Φ to offloading strategy X, e.g., we just need to decide the offloading strategy X, then the partition strategy P and scheduling strategy Φ will be decided accordingly. As a result, our initial optimization problem *POSP* can be transformed into an 0-1 integer optimization problem which only has one set of variables X:

$$\begin{aligned} \mathcal{P}: \quad & \min_{X} T \\ s.t. \quad & C1: \ x_{ij} = 1 \xrightarrow{yields} p_j = p_{ij}, \\ & C2: \ \sum_{i=1}^{m} x_{ij} \le 1, \forall\ 1 \le j \le n, \\ & C3: x_{ij} = 1 \ or \ 0, \forall\ 1 \le j \le n, \forall\ 1 \le i \le m, \\ & C4: \phi(s_i) = \{j |\ x_{ij} = 1\}. \end{aligned} \tag{14}$$

Actually, this is a variant of general assignment problem (GAP) where the cost of assigning a task to a server, i.e., its weighted inference latency $w_j T_{ij}$, can be changed by the other tasks assigned to the same server since the waiting time t_{ij}^{wait} is affected by X. Motivated by the algorithm proposed by Ross, G. Terry, and Richard M. Soland [20], we design a heuristic algorithm BBO, i.e., branch and bound optimization to solve this problem. The detailed introduction of BBO is in Section **4.2**. Once the offloading strategy X is determined by BBO, partition strategy P and scheduling strategy Φ can be determined accordingly, thus determining the solution of the initial problem *POSP*.

4.2 Branch and Bound Optimization Algorithm (BBO)

Actually, branch and bound is a way to traverse the entire solution space of the problem with pruning to limit the time complexity. Branch is for generating the solution and bound is for pruning. According to branch and bound, the solution set for $\mathcal{P}$ is separated into two mutually exclusive and collectively exhaustive subsets based on the 0-1 dichotomy of variable values, so as to the subsets created. **Fig. 7** gives an example of separating the solution set of $\mathcal{P}$. Each separation creates two new candidate problems whose solution sets differ only in the value assigned to a particular variable. We use BFS (breadth first search) to traverse the solution sets with bounding and pruning.

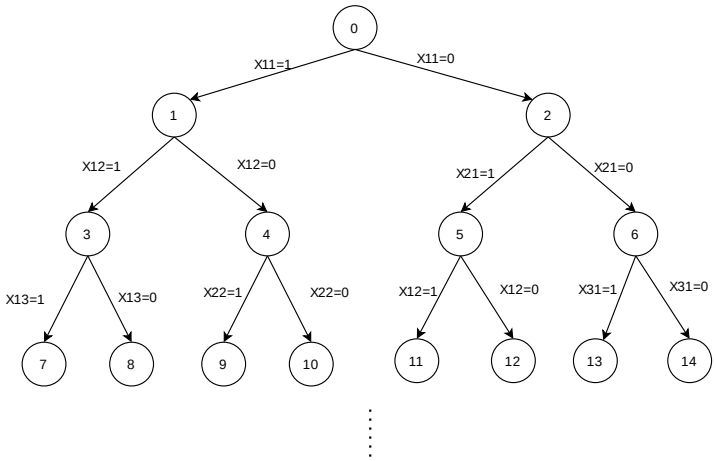


Fig. 7. An example of the solution space of this problem

The main processing procedures for each candidate problem $\mathcal{P}_k$ are:

1) **Bounding**: make a relaxation of $\mathcal{P}_k$ to get a lower bound Θ_k of the objective function in this branch according to the relaxed problem $\mathcal{PR}_k$ and update the upper bound of the objective function for problem $\mathcal{P}$ by substituting a feasible solution into the objective function.

2) **Branching**: select a variable as the separation variable to further separate the solution set.

3) **Pruning**: check if the lower bound of this branch is too large that needs to be pruned. These procedures' detailed explanations are as follows, and for notational convenience, let $\mathcal{P}_k$ denote the candidate problem of a branch and F_i denote the set of tasks which has been fixed to be offloaded to ES s_i in $\mathcal{P}_k$.

4.2.1 Bounding

Bounding is the procedure to bound the upper bound and lower bound of each candidate problem for pruning. Relaxing the problem to get the bound in a traditional method in branch and bound algorithm. For the current candidate problem $\mathcal{P}_k$, it is too complex to solve since the assignment cost of each task is tightly relate to the offloading strategy of other tasks, e.g., the assignment cost of each task is unknown at the beginning. Thus, we can get the relaxed problem $\mathcal{PR}_k$ by fixing the assignment of each task relative to each ES before solving the problem. In the current candidate problem $\mathcal{P}_k$, some tasks' assignment has been decided. Therefore, we can first compute the total weighted inference latency of an ES s_i only considering the tasks have been decided to be offload to it. Then, for each task j whose assignment has not been decided, the assignment cost C_{ij} equals to the increment of the total weighted inference latency of s_i after assigning task j to s_i:

$$C_{ij} = T_i(j) - T_i\,, \tag{15}$$

where T_i denotes the existing total weighted inference latency of tasks in F_i utilizing SWRTF and $T_i(j)$ is the new total weighted inference latency of tasks in $F_i \cup \{j\}$ utilizing SWRTF. The relaxed problem $\mathcal{PR}_k$ can be formalized as:

$$\begin{aligned} \mathcal{PR}_k: \quad & \min_{X} T' \\ s.t. \quad & C1: \sum_{i=1}^{m} x_{ij} \le 1, \forall\, 1 \le j \le n, \\ & C2: \; x_{ij} = 1 \; or \; 0, \forall\, 1 \le j \le n, \forall\, 1 \le i \le m, \end{aligned} \tag{16}$$

where $T' = \frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}C_{ij}$ is the average weighted assignment cost. Since C_{ij} is known,

problem $\mathcal{PR}_k$ has an obvious solution X_k by assigning every unassigned task $j \in \bigcup_{i=1}^{m} F_i$ to the ES s_{i_j} that minimizes C_{ij}. Substituting this solution into $\mathcal{PR}_k$ yields the lower bound of this branch, denoted by Θ_k. It is obvious that X_k is a feasible solution for $\mathcal{P}_k$, so we can calculate a valid objective function value T_k^{valid} of the initial problem $\mathcal{P}$ by substituting X_k into $\mathcal{P}_k$. Since the problem seeks for minimum objective function value, T_k^{valid} is also the upper bound Ω of $\mathcal{P}$.

Function $ProblemRelax(\mathcal{P}_k)$
$\Theta_k \leftarrow 0$, $J_k \leftarrow$ tasks that have not been determined to be offloaded to which ES in $\mathcal{P}_k$;
for i from 1 to m do
  $F_i \leftarrow$ set of tasks have been determined to be offloaded to s_i;
  $T_i \leftarrow$ the total weighted inference latency of F_i according to *SWRTF*;
  $\Theta_k \leftarrow \Theta_k + T_i$;
foreach $j \in J_k$ do
  for i from 1 to m do
    $T_i(j) \leftarrow$ the total weighted inference latency of $F_i \cup \{j\}$ according to *SWRTF*;
    $C_{ij} \leftarrow T_i(j) - T_i$;
foreach $j \in J_k$ do
  $i^* \leftarrow argmin_i\{C_{ij}\}$;
  for i from 1 to m do
    $x_{ij} \leftarrow false$;
  $x_{i^*j} \leftarrow true$;
  $\Theta_k \leftarrow \Theta_k + T_{i^*j}$;
return X, Θ_k

4.2.2 Branching

Branching is the procedure to separate the solution space of the problem into different sub-problems for traversing. To select a $x_{ij} \in X$ to be the separation variable, i.e., separate the problem according to the value of x_{ij}, we compute a "re-offloading profit" δ_{ij} for each ES s_i and each task j that has not been determined to be offloaded to which ES in the current candidate problem, that is, $j \notin \bigcup_{i=1}^{m} F_i$. Let $X_k(i)$ denote the solution by modifying the feasible solution X_k with re-offloading task j to ES s_i. It is obvious that $X_k(i)$ is also a feasible solution for $\mathcal{P}_k$. Then δ_{ij} can be defined as the reduction of the objective function T in $\mathcal{P}_k$:

$$\delta_{ij} = T(X_k) - T(X_k(i)), \tag{17}$$

where $T(X_k)$ denotes the value of objective function by substituting X_k into problem $\mathcal{P}_k$. The selected separation variable $x_{i^*j^*}$ is the one that has maximum $\delta_{i^*j^*}$ among those with $j \notin \bigcup_{i=1}^{m} F_i$ in the candidate problem $\mathcal{P}_k$. Then, the solution set is further separated into two subsets, one with $x_{i^*j^*} = 1$ and the other one with $x_{i^*j^*} = 0$.

Function $MaxProfit(\mathcal{P}_k, X_k)$
$J_k \leftarrow$ tasks that have not been determined to be offloaded to which ES in $\mathcal{P}_k$;
$T(X_k) \leftarrow$ the value of objective function by substituting X_k into problem $\mathcal{P}_k$;
foreach $j \in J_k$ do
  for i from 1 to m do
    $X_k(i) \leftarrow X_k$;
    for l from 1 to m and $l \neq i$ do
      $x_{lj} \leftarrow 0$; //here x_{lj} represents the variable in $X_k(i)$
    $x_{ij} \leftarrow 1$; //here x_{ij} represents the variable in $X_k(i)$

9: $T(X_k(i)) \leftarrow$ the value of objective function by substituting $X_k(i)$ into problem $\mathcal{P}_k$;
10: $\delta_{ij} = T(X_k) - T(X_k(i))$;
19: $i, j \leftarrow argmax_{i,j}\{\delta_{ij}\}$;
20: return i, j

4.2.3 Pruning

Pruning is the procedure to limit the time complexity of the traverse of solution space. To avoid redundant computing, we safely discard some solution set based on two rules. First, as described in the bounding procedure, we can maintain the upper bound Ω of the initial problem $\mathcal{P}$ and compute a lower bound Θ_k for each candidate problem $\mathcal{P}_k$. If $\Theta_k > \Omega$, this branch should be pruned obviously since it is impossible to generate the optimal solution. Then, during the BFS of the solution sets, we set a threshold ω of the maximum number of candidate problems in each level. Particularly, the candidate problems in the same level are processed in a round and if the number of candidate problems in a round exceeds ω, those with maximum lower bound Θ_k are pruned.

4.2.4 Algorithm Design and Analysis

Algorithm 2 illustrates the pseudocode of our BBO algorithm. We maintain two problem sets Q and Q' for BFS and an upper bound Ω of the initial problem $\mathcal{P}$ (line 1). The main loop of this algorithm is the BFS process of the solution sets and each loop can be decomposed into three steps. In step1, we compute the lower bound Θ_k and feasible solution X_k of the candidate problem $\mathcal{P}_k$ utilizing a function $ProblemRelax(\mathcal{P}_k)$ (line 4). In step2, if the lower bound Θ_k of $\mathcal{P}_k$ is not greater than Ω, we update the upper bound Ω (lines 6-7) and select a separation variable $x_{i^*j^*}$ for branching with the function $MaxProfit(\mathcal{P}_k, \mathrm{X}_k)$ (lines 8-11). After conducting step1 and step2 for each candidate problem in this round (line 3), we check if the problem set for next round is too big and remove some candidate problems if necessary (lines 12-14). At last, we return the solution with minimum objective function value (lines 21-22). There are mn processing rounds and we need to process at most ω candidate problems each round and the time complexity of each candidate problem is $O(n^2 \log n)$. Thus, the time complexity of BBO algorithm is $O(\omega mn^3 \log n)$.

Algorithm 2. BBO (Branch and Bound Optimization)
Input: The EDs set D, the ESs set $S = \{s_1, s_2, \cdots, s_m\}$, the initial problem $\mathcal{P}$
1: Initialize the problem set of this round $Q \leftarrow \{\mathcal{P}\}$ and the problem set of next round $Q' \leftarrow \emptyset$, upper bound $\Omega \leftarrow \infty$;
2: for z from 1 to mn do
3: foreach $\mathcal{P}_k \in Q$ do
4: $[\Theta_k, \mathrm{X}_k] \leftarrow ProblemRelax(\mathcal{P}_k)$;
5: if $\Theta_k \leq \Omega$ then
6: $T_k^{valid} \leftarrow \mathcal{P}_k(\mathrm{X}_k)$;
7: $\Omega \leftarrow \min\{\Omega, T_k^{valid}\}$;
8: $[i^*, j^*] \leftarrow MaxProfit(\mathcal{P}_k, \mathrm{X}_k)$;
9: $\mathcal{P}_x \leftarrow \mathcal{P}_k$ with $X_{i^*j^*} = 1$;
10: $\mathcal{P}_y \leftarrow \mathcal{P}_k$ with $X_{i^*j^*} = 0$;
11: $Q' \leftarrow Q' \cup \{\mathcal{P}_x, \mathcal{P}_y\}$;
12: while $|Q'| > \omega$ do
13: $\mathcal{P}_r \leftarrow$ the problem with max lower bound Θ_r in Q';
14: $Q' \leftarrow Q' \backslash \{\mathcal{P}_r\}$;

15: $Q \leftarrow Q'$;
16: $Q' \leftarrow \emptyset$;
17: $\mathcal{P}_t \leftarrow argmin_{\mathcal{P}_t \in Q}\{T_t^{valid}\}$;
18: $X_t \leftarrow$ the solution of $\mathcal{P}_t$;
19: return X_t;

Since BBO gives the offloading strategy X, the partition strategy P and scheduling strategy Φ can be given accordingly based on **Section 4.1**.

5. Implementation and Evaluation

In this section, we first introduce the prototype setup for our experiment, and then compare our scheme with several existing schemes.

5.1 Prototype Setup

To evaluate the performance of our scheme, we build a heterogeneous device-edge system prototype. We use two Laptops to act as the edge servers to assist the end devices in executing their inference tasks, one equipped with a 6-core 2.60GHz Intel CPU and16-GB RAM and the other one equipped with a 4-core 1.60GHz Intel CPU and 8-GB RAM. The end devices are composed of two Raspberry Pis, each of them equipped with a 4-core ARM Cortex-A72 CPU, a JetsonTX2 equipped with a 4-core ARM Cortex-A57 CPU, a Jetson xavierNX equipped with a 6-core ARM V8 CPU. All end devices are connected to the edge server through LAN. We use the pretrained DNN models from the standard implementation from famous package PyTorch. An AlexNet model is deployed on Raspberry Pi1, a MobileNet_V2 model is deployed on Raspberry Pi2, a ResNe18 model is deployed on the JetsonTX2 and a VGG19 model is deployed on the Jetson xavierNX. Let the priority of the tasks on the four end devices be 1, 2, 3, and 4, respectively. The settings are listed in **Table 2**. We use tiny-ImageNet [21], a subset of the ILSVRC2012 classification dataset, as our dataset.

Table 2. Experimental Settings

Symbol	Device Info	Inference model	Priority
Pi4B1	Raspberry Pi 4B with a 4-core ARM Cortex-A72 CPU	AlexNet	1
Pi4B2	Raspberry Pi 4B with a 4-core ARM Cortex-A72 CPU	MobileNet-V2	2
Jetson-TX2	JetsonTX2 with a 4-core ARM Cortex-A57 CPU	ResNet18	3
Jetson-NX	Jetson xavierNX with a 6-core ARM V8 CPU	VGG19	4
Laptop-6c16g	Laptop with a 6-core 2.60GHz Intel CPU and16-GB RAM		
Laptop-4c8g	Laptop with a 4-core 1.60GHz Intel CPU and 8-GB RAM		

5.2 Benchmarks

We evaluate our scheme by comparing the performance with four naive schemes and a SOTA (state-of-the-art) scheme as follows:

- **Local-Only (LO):** All inference tasks are executed locally.
- **Edge-Only (EO):** All EDs offload their tasks to the edge servers without DNN partition, the selection of the edge server to offload is random and the execution order of tasks on the server is FCFS.
- **DADS** [7] **with random allocation and FCFS scheduling (RA-FS):** We first use DADS to compute the optimal partition of each task for each ES, and then randomly allocate tasks to ESs where tasks are scheduled in the FCFS manner.
- **DADS with random allocation and SWTRF scheduling (RA):** We first use DADS to compute the optimal partition of each task for each ES, and then randomly allocate tasks to ESs where tasks are scheduled in the SWTRF manner.
- **CCORAO** [22]**:** This is an SOTA (state-of-the-art) algorithm for cloud assisted mobile edge computing in vehicular networks. The main idea of **CCORA** is to decide the offloading strategy and resource allocation strategy iteratively. We modify it to our problem *POSP*, i.e., deciding the offloading strategy and scheduling strategy iteratively.

5.3 Experimental Results

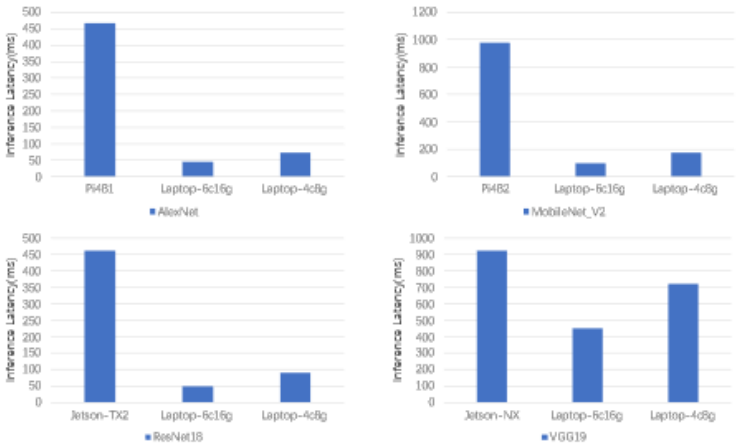


Fig. 8. The inference latency of different models on different devices.

We first test the inference latency of each task on different devices. It can be seen in **Fig. 8** that the computing capacity of different devices varies and the inference latency when tasks are executed locally is too high to support some real-time applications. Thus, we need to carefully design the collaborative inference scheme to reduce the inference latency in this heterogeneous system.

Then we conduct multiple experiments by modifying the bandwidth between devices. The bandwidth configurations for the experiments are shown in **Table 3**. We use the four different bandwidth configurations to simulate different network conditions of the system. Configuration1 simulates the situation that the overall network condition of the system is good, Configuration2 simulates the situation that the overall network condition of the system is bad, Configuration3 simulates the situation that the network condition between different devices varies and Configuration4 simulates the situation that the network conditions between all end devices to the 2 ESs has a large gap.

Table 3. Four bandwidth configurations

Bandwidth Configuration1				Bandwidth Configuration2		
	Laptop-6c16g	Laptop-4c8g			Laptop-6c16g	Laptop-4c8g
Pi4B1	5.0 Mbps	5.0 Mbps		Pi4B1	0.5 Mbps	0.5 Mbps
Pi4B2	5.0 Mbps	5.0 Mbps		Pi4B2	0.5 Mbps	0.5 Mbps
Jetson-TX2	5.0 Mbps	5.0 Mbps		Jetson-TX2	0.5 Mbps	0.5 Mbps
Jetson-NX	5.0 Mbps	5.0 Mbps		Jetson-NX	0.5 Mbps	0.5 Mbps
Bandwidth Configuration3				Bandwidth Configuration4		
	Laptop-6c16g	Laptop-4c8g			Laptop-6c16g	Laptop-4c8g
Pi4B1	1.0 Mbps	0.5 Mbps		Pi4B1	0.1 Mbps	5.0 Mbps
Pi4B2	0.7 Mbps	1.2 Mbps		Pi4B2	0.1 Mbps	5.0 Mbps
Jetson-TX2	1.2 Mbps	0.5 Mbps		Jetson-TX2	0.1 Mbps	5.0 Mbps
Jetson-NX	0.3 Mbps	2.9 Mbps		Jetson-NX	0.1 Mbps	5.0 Mbps

Table 4 shows the partition and offloading strategies for tasks given by CIS at different configurations, where 2/11 means the model AlexNet has 11 layers and is partitioned at the 2nd layer, 0/11 means offloading the entire task to the edge server and 11/11 means executing the entire task locally. We can draw from the results that the better the network condition, the earlier the tasks are offloaded. When the system suffers adverse network conditions, the end devices tend to execute their tasks locally.

Table 4. Partition and offloading strategy at different configurations.

Bandwidth Configuration1				Bandwidth Configuration2		
	Partition	Offloading			Partition	Offloading
Pi4B1	2/11	s_1		Pi4B1	11/11	s_1
d_2	0/55	s_2		d_2	22/55	s_2
d_3	0/18	s_1		d_3	18/18	s_1
d_4	0/22	s_1		d_4	22/22	s_2
Bandwidth Configuration3				Bandwidth Configuration4		
	Partition	Offloading			Partition	Offloading
Pi4B1	2/11	s_1		Pi4B1	2/11	s_1
d_2	22/55	s_1		d_2	0/55	s_2
d_3	18/18	s_1		d_3	0/18	s_1
d_4	0/22	s_2		d_4	0/22	s_1

Fig. 9 shows the average weighted inference latency of different schemes. It can be seen that our scheme CIS has the similar performance to CCORAO, both of which can reduce about 29% to 71% on the average weighted inference latency compared to the other four naïve schemes. Further analysis of the experimental results reveals that the result of LO is stable but can't be optimal, the result of RO is influenced by network conditions, when the network is poor, it may lead to intolerable latency, and the results of RA-FS and RA are relatively better since they make some improvement to LO and RO. CIS and CCORAO jointly considers DNN partition, task offloading and scheduling, so they can always obtain the optimal solutions when the problem space is small.

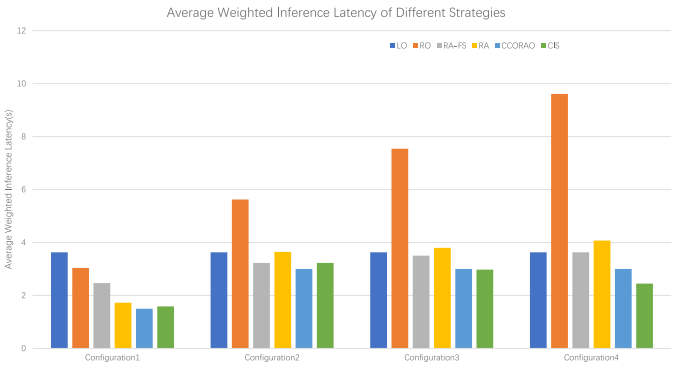


Fig. 9. The average weighted inference latency of different schemes at different configurations.

5.4 Simulation Experiments

In this section, a series of simulation experiments are conducted to further evaluate the performance of our proposed scheme CIS. Initially, we evaluate the performance of CIS with different network conditions in Section **5.4.1**. Then we test the schemes when the size and number of tasks change in Section **5.4.2**. At last, the robustness of CIS with different computational capacities patterns for the multiple edge servers and devices is validated in Section **5.4.3**. For the numerical analysis, the computational capacities of EDs, the computational capacities of ESs and the bandwidth between ED to ES take a uniform distribution in the range of [1, 5] FLOPS, [10, 20] FLOPS and [0.1, 2.0] Mbps, respectively. The DNN models in our experiments include: AlexNet, MobileNet_V2, ResNet18 and VGG19.

5.4.1 Performance with different network conditions

In this section, we evaluate the performance of CIS with different network conditions. The simulation scenario has 12 EDs and 6 ESs, the computational capacities of EDs are: [1.2, 1.4, 1.6, 1.8, 2, 2.2, 2.4, 2.6, 2.8, 2.9, 2.3, 4.0], the computational capacities of ESs are: [11, 15, 17.5, 20, 24, 22], the priorities of tasks on the EDs are: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. An AlexNet model is deployed d_1, d_5, d_9, a MobileNet_V2 model is deployed d_2, d_6, d_{10}, a ResNet18 model is deployed d_3, d_7, d_{11}, and a VGG19 model is deployed d_4, d_8, d_{12}. For convenience, we set the bandwidth between all devices the same. **Fig. 10** shows the simulation results at difference bandwidth. The inference latency of CIS and CCORAO are similar and always smaller than that of other schemes since the problem space is small and they can always get the optimal solution. The inference latency of LO still does not vary with bandwidth and with the continuous improvement of network bandwidth, the performances of other five schemes are also improving. At the beginning, the improvement is quite significant since the network condition is the main bottleneck at this time. When the bandwidth is high enough, this improvement gets smaller as the computational capacity is the main bottleneck at this time. Overall, our proposed scheme CIS has a well performance at different network conditions.

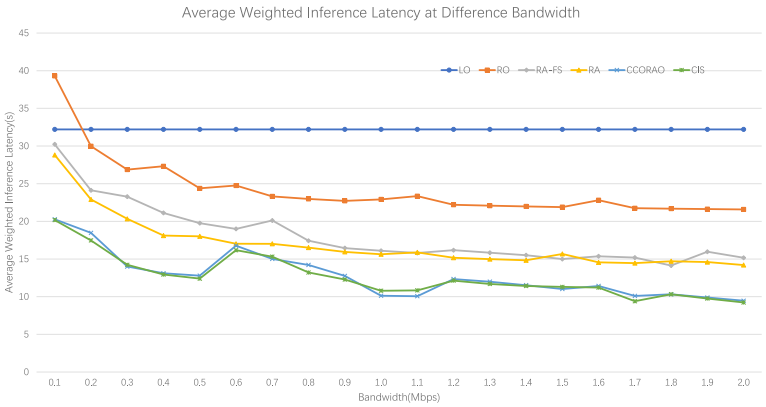


Fig. 10. Average weighted inference latency at difference bandwidth.

5.4.2 Performance with different numbers of tasks

In this section, we evaluate the performance of CIS with different numbers of tasks. We fix 100 ESs whose computational capacities are randomly taken from [10, 20] FLOPS. Then we conduct a series of experiments with different number of EDs, e.g., different number of tasks. From each number of EDs, we conduct the experiments for 50 times. For each experiment, the computational capacities of the EDs are randomly taken from [1, 5] FLOPS, the DNN model deployed on it is randomly picked from AlexNet, MobileNet_V2, ResNet18 and VGG19 and the bandwidth between an ED and an ES is randomly taken from [0.1, 2.0] Mbps. We take the results of scheme LO as the baseline and compute the relative average weighted inference latency for other five schemes, e.g., the average weighted inference latency of other scheme divided by that of LO. Then the average of results of the 50 times experiments is taken to compare different schemes. **Fig. 11** shows the simulation results. There are four main observations regarding these results:

- The performance of each scheme decreases as the number of tasks increases. This is intuitive since the number and computational capacities of the ESs are fixed.
- RO is always the worst scheme and can lead to more than 140% average weighted inference latency compared to LO. This is because we randomly select the ES to offload for each task, which means many tasks may be offloaded to the same ES, thus increasing the result. Likely, RA-FS and RA can also have worse performance compared to LO since the selection of ESs to offload is random. They are better than RO since they will first partition the DNN models, which can reduce the computation overhead of the ESs.
- The performance gap between CIS and RA-FS or RA increases as the number of tasks increases, because the number of tasks offloaded to the same ES is small at the beginning, that is, the scheduling strategy of tasks has little impact on the results. The more the tasks, the more important it is to decide proper offloading strategy and scheduling strategy.
- Compared to CCORAO, when the number of tasks is small, the advantage of CIS is not obvious since they can both get the optimal solutions. However, as the number of tasks increases, CCORAO cannot conduct enough rounds of iteration to get convergence. Although CIS also discard some subproblems to limit its complexity, our careful design of heuristic rules for discarding subproblems does have an undeniable impact on the results.

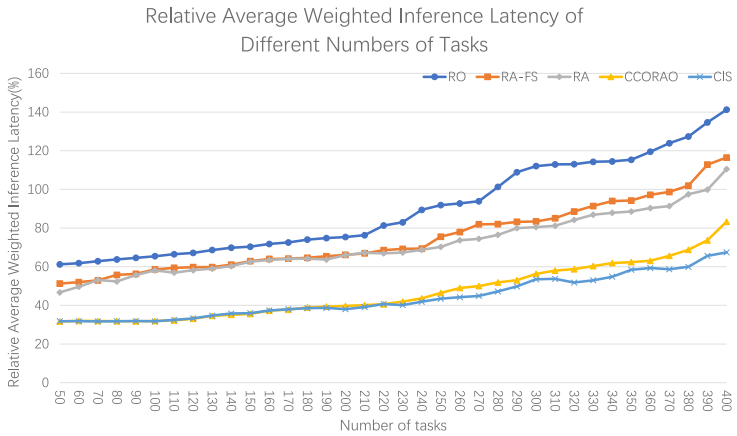


Fig. 11. Relative average weighted inference latency of different numbers of Tasks

5.4.3 Performance with different computational capacities patterns

In this section, we validate the robustness of CIS with different computational capacities patterns for the multiple edge servers and devices. We fix the number of EDs and ESs 300 and 100 respectively. For each DNN model, there are 75 EDs deployed with it. The computational capacities of ESs and EDs are randomly taken from [10, 20] FLOPS and [1, 5] FLOPS respectively. We conduct the experiments for 100 times and compare the relative average weighted inference latency for other five schemes. The results are shown in **Fig. 12**. Compared with RO, RA-FS and RA, the performance our scheme CIS has obvious advantages. Overall, the inference latency when taking CIS is always lower. What's more, the results of CIS are less dispersed, which means unacceptable results rarely occur. This is mainly due to the fact that the first three strategies randomly select the offloading strategy, which is obviously not feasible when the number of tasks is large. When it comes to CCORAO and CIS, both of them almost always demonstrates a certain level of performance improvement compared to the baseline scheme LO. However, the results of CIS exhibit a more concentrated distribution and a lower highest inference latency. This indicates our carefully designed heuristic rules for discarding subproblems in Section **4.2.3** do make sense.

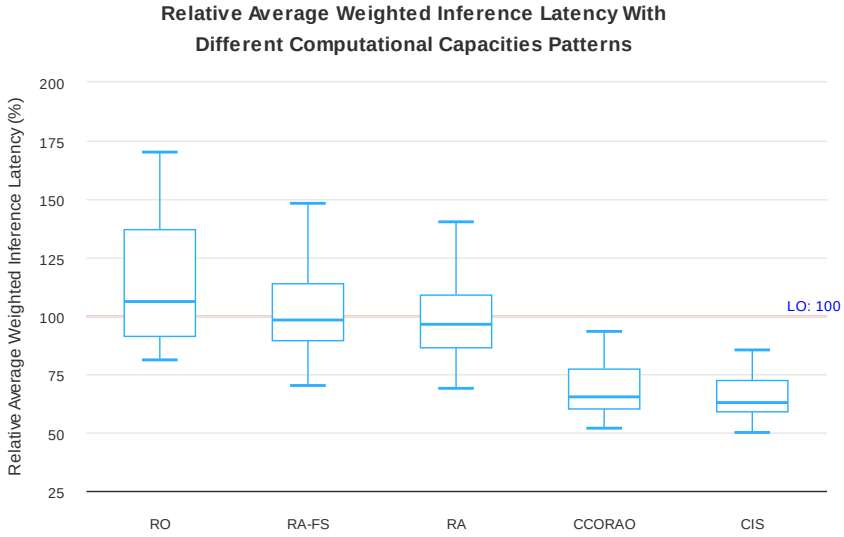


Fig. 12. Relative average weighted inference latency with different computational capacities patterns

6. Conclusion

In this paper, we study the DNN inference acceleration in a heterogeneous edge computing scenario. We present a comprehensive analysis of the collaborative inference in the heterogeneous scenario and point out the complexity of this problem. A scheme CIS is proposed which jointly combines DNN partition, task offloading and task scheduling to accelerate the DNN inference tasks. Extensive experiments are conducted to evaluate our scheme. With a detailed analysis of evaluation results, CIS are validated to be more effective for improving the average weighted inference latency of the system.

Acknowledgement

This work was supported in part by the Science and Technology Project of State Grid Co., LTD (Research on data aggregation and dynamic interaction technology of enterprise-level real-time measurement data center, 5108-202218280A-2-399-XG).

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol.521, pp.436-444, May, 2015. Article (CrossRef Link)

[2] J. Chen, and X. Ran, "Deep Learning With Edge Computing: A Review," in *Proc. of Proceedings of the IEEE*, vol.107, no.8, pp.1655-1674, Aug. 2019. Article (CrossRef Link)

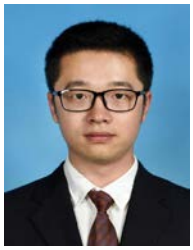
[3] J. Chai, and A. Li, "Deep Learning in Natural Language Processing: A State-of-the-Art Survey," in *Proc. of 2019 International Conference on Machine Learning and Cybernetics (ICMLC)*, pp. 1-6, 2019. Article (CrossRef Link)

[4] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Proc. of 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp.4960-4964, 2016. Article (CrossRef Link)

[5] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "MoDNN: Local distributed mobile computing system for Deep Neural Network," in *Proc. of Design, Automation & Test in Europe Conference & Exhibition*, pp.1396-1401, 2017. Article (CrossRef Link)
[6] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low Latency Geo-distributed Data Analytics," *ACM SIGCOMM Computer Communication Review*, vol.45, no.4, pp.421-434, Aug. 2015. Article (CrossRef Link)
[7] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge," in *Proc. of IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp.1423-1431, 2019. Article (CrossRef Link)
[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *Proc. of 2015 lEEE Conference on Computer Vision and Pattern Recognition*, pp.l-9, 2015. Article (CrossRef Link)
[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, pp.770-778, 2016. Article (CrossRef Link)
[10] M. Gao, R. Shen, L. Shi, W. Qi, J. Li, and Y. Li, "Task Partitioning and Offloading in DNN-Task Enabled Mobile Edge Computing Networks," *IEEE Transactions on Mobile Computing*, vol.22, no.4, pp.2435-2445, Apr. 2023. Article (CrossRef Link)
[11] X. Tang, X. Chen, L. Zeng, S. Yu, and L. Chen, "Joint Multiuser DNN Partitioning and Computational Resource Allocation for Collaborative Edge Intelligence," *IEEE Internet of Things Journal*, vol.8, no.12, pp.9511-9522, 2021. Article (CrossRef Link)
[12] T. Mohammed, C. Joe-Wong, R. Babbar, and M. Di Francesco, "Distributed Inference Acceleration with Adaptive DNN Partitioning and Offloading," in *Proc. of IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp.854-863, 2020. Article (CrossRef Link)
[13] C.-Y. Yang, J.-J. Kuo, J.-P. Sheu, and K.-J. Zheng, "Cooperative Distributed Deep Neural Network Deployment with Edge Computing," in *Proc. of ICC 2021 - IEEE International Conference on Communications*, pp.1-6, 2021. Article (CrossRef Link)
[14] Z. Liao, W. Hu, J. Huang, and J. Wang, "Joint multi-user DNN partitioning and task offloading in mobile edge computing," *Ad Hoc Networks*, vol.144, 2023. Article (CrossRef Link)
[15] L. Shi, Z. Xu, Y. Sun, Y. Shi, Y. Fan, and X. Ding, "A DNN inference acceleration algorithm combining model partition and task allocation in heterogeneous edge computing system," *Peer-to-Peer Networking and Applications*, vol.14, pp.4031-4045, 2021. Article (CrossRef Link)
[16] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge," *ACM SIGARCH Computer Architecture News*, vol.45, no.1, pp.615-629, 2017. Article (CrossRef Link)
[17] S. Zhang, Y. Li, X. Liu, S. Guo, W. Wang, J. Wang, B. Ding, and D. Wu, "Towards Real-time Cooperative Deep Inference over the Cloud and Edge End Devices," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol.4, no.2, pp.1-24, Jun. 2020. Article (CrossRef Link)
[18] N. Wang, Y. Duan, and J. Wu, "Accelerate Cooperative Deep Inference via Layer-wise Processing Schedule Optimization," in *Proc. of 2021 International Conference on Computer Communications and Networks*, pp.1-9, 2021. Article (CrossRef Link)
[19] Y. Duan, and J. Wu, "Joint Optimization of DNN Partition and Scheduling for Mobile Cloud Computing," in *Proc. of ICPP '21: Proceedings of the 50th International Conference on Parallel Processing*, pp.1-10, 2021. Article (CrossRef Link)
[20] G. T. Ross, and R. M. Soland, "A branch and bound algorithm for the generalized assignment problem," *Mathematical programming*, vol.8, pp.91-103, Dec. 1975. Article (CrossRef Link)
[21] Y. Le, and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231N*, vol.7, no.7, 2015. https://cs231n.stanford.edu/reports/2015/pdfs/yle_project.pdf
[22] J. Zhao, Q. Li, Y. Gong and K. Zhang, "Computation Offloading and Resource Allocation for Cloud Assisted Mobile Edge Computing in Vehicular Networks," *IEEE Transactions on Vehicular Technology*, vol.68, no.8, pp.7944-7956, 2019. Article (CrossRef Link)

[23] Y. Chen, K. Wu, Q. Zhang, “From QoS to QoE: A Tutorial on Video Quality Assessment”, *IEEE Communications Surveys & Tutorials*, vol.17, no.2, pp.1126-1165, 2015. Article (CrossRef Link)

Meizhao Liu received her PhD degree in Power System and Automation from Zhejiang University, Hangzhou, China, in 2009. Currently, she is a core member of the Data Operation Management Department at the State Grid Jiangsu Electric Power Co., Ltd. Her research interests include data modeling, big data, and task scheduling. To date, she has published six papers in refereed journals and conferences and received several provincial and ministerial awards for science and technology innovation.



Yingcheng Gu received his master's degree in Software Engineering from Xi'an Jiaotong University, Xi'an, China, in 2018. He is currently working on big data processing and analysis at the State Grid Jiangsu Electric Power Co., Ltd. Information \& Telecommunication Branch. His main research interests include big data processing and artificial intelligence.

Sen Dong received the BS degree from the Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2023, where he is currently working towards the MS degree under the supervision of Associate Professor Sheng Zhang. He is a member of the State Key Laboratory for Novel Software Technology. Currently, his research interests include edge computing and collaborative inference.

Liu Wei received her master's degree in Computer Engineering from Peking University, Beijing, China, in 2021. She is now working at the State Grid Jiangsu Electric Power Co., Ltd. Information & Telecommunic.

Kai Liu received his master's degree in Computer Technology from Sun Yat-sen University, Guangzhou, China, in 2018. He is currently working on data management and big data analysis at the State Grid Jiangsu Electric Power Co., Ltd. Information & Telecommunication Branch. His main research interests include big data modeling and application. To date, he has published nine papers in refereed journals and conferences.

Yuting Yan received the BS degree in financial engineering at Business School, Nanjing University, China, in 2021. She is currently working toward the PhD degree in the Department of Computer Science and Technology, Nanjing University, under the supervision of associate professor Sheng Zhang. She is a member of the State Key Laboratory for Novel Software Technology. She has published 10 papers on conferences or journals including IEEE SECON, IEEE INFOCOM and TMC. Currently, her research interests include mobile edge intelligence and video analytics.

Yu Song received his master's degree in Software Engineering from Southeast University, Nanjing, China, in 2019. He is a member of the Data Operation Management Department at the State Grid Jiangsu Electric Power Co., Ltd. Information & Telecommunication Branch. His research interests include machine learning and big data analytics. To date, he has published four papers, including those appeared in Knowledge Based Systems and IJCAI.

Huanyu Cheng received his bachelor's degree in Computer Science and Technology from Wuhan University, Wuhan, China. He is now working on cloud computing, big data processing, and artificial intelligence research at the State Grid Jiangsu Electric Power Co., Ltd. Information & Telecommunication Branch. To date, he has published three papers in refereed journals and conferences.

Lei Tang received his bachelor's degree in Computer Science and Technology from the Department of Computer Engineering, Southeast University, Nanjing, China, in 2000, and received his master's degree in Power System and Automation from Nanjing University of Technology, Nanjing, China, in 2009. He is a senior engineer and his main research interests include data analytics, data modeling, domain-wide data management, and customer behavior research.

Sheng Zhang is an associate professor with Nanjing University, and a member of the State Key Lab. for Novel Software Technology. He received the BS and PhD degrees from Nanjing University. His current research interests include edge computing and edge intelligence. He regularly publishes in scholarly journals and conference proceedings, such as JSAC, TMC, TPDS, TC, MobiHoc, ICDCS, and INFOCOM. He is the recipient of CCFSys Best Paper Award (2023), IEEE ICPADS Outstanding Paper Runner-Up Award (2021), IEEE ICCCN Best Paper Award (2020), IEEE MASS Best Paper Runner-Up Award (2012), ACM Nanjing Rising Star (2020), ACM China Doctoral Dissertation Nomination Award (2015). He is a senior member of IEEE and CCF, and a member of ACM.