POLITECNICO DI TORINO
Repository ISTITUZIONALE

Towards Energy-Efficient Collaborative Inference and Fine-tuning: Matching Model Compression and Offloading with Resource Availability

Original
Towards Energy-Efficient Collaborative Inference and Fine-tuning: Matching Model Compression and Offloading with Resource Availability / Zhou, Y., Ma, L., Wang, X., Li, Q., Chiasserini, C.F., Han, G.. - In: IEEE TRANSACTIONS ON NETWORKING. - ISSN 2998-4157. - 34:(2026), pp. 5127-5142. [10.1109/TON.2026.3689748]

Availability:
This version is available at: 11583/3010272 since: 2026-05-06T09:30:36Z

Publisher:
IEEE

Published
DOI:10.1109/TON.2026.3689748

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

09 November 2026

Towards Energy-Efficient Collaborative Inference and Fine-tuning: Matching Model Compression and Offloading with Resource Availability

Yue-e Zhou, Lianbo Ma*, *Senior Member, IEEE,* Xingwei Wang, Qing Li, *Senior Member, IEEE,* Carla Fabiana Chiasserini, *Fellow, IEEE*, Guangjie Han, *Fellow, IEEE*

***Abstract*—We consider the collaborative inference acceleration task via cloud-edge-end collaboration, which involves a series of tightly coupled decision-making steps, including *which* DNN model to be selected, *how much* to compress model, *how* to partition model, and *where* to offload partitioned submodels. In practical deployments, these decisions jointly affect both fine-tuning and inference performance, and must jointly account for such aspects as the model being used, the computational resources and local datasets available at each device, as well as network latencies, which significantly increases the complexity of optimizing the problem. Yet, no existing studies focus on such joint optimization problem for these tightly coupled decisions. In this paper, we model this problem as a multi-dimensional optimization problem, jointly optimizing collaborative inference and fine-tuning by selecting the DNN model, compression level, partition strategy, and computational resource allocation, with the objective of minimizing the overall energy consumption of the learning-inference process, subject to accuracy and latency constraints. To this end, we propose an algorithmic framework called JQODI combining a time-energy tree diagram to represent the learning process, a dynamic programming solution strategy, and a data-driven theoretical approach to predict the expected total number of training epochs that meet the accuracy requirements. We prove that JQODI approximates the optimal solution with polynomial complexity. Numerical results demonstrate that JQODI surpasses state-of-the-art methods in both energy efficiency and latency.**

***Index Terms*—Cloud-edge-end collaboration, edge computing, distributed learning, quantization and partition.**

I. Introduction

With the fast growth in data being generated at edge network, advanced edge devices (e.g., IoT devices) are expected to execute machine learning (ML) (e.g., deep neural network (DNN)) models for various real-time edge computing (EC) applications [1], [2], [4]. In EC environments, these edge devices suffer from limitations of computation resources with stringent requirements of energy consumption [3], [5], while training DNNs requires large quantities of computational resources. Examples include smart city and smart factory ML-based applications [6]–[9], where a series of DNN models need to be deployed on target end devices/smart devices (i.e., IoT devices, or smart mobile devices) (SDs) [10].

Example 1: *In the smart city system, as shown in Figure 1, many SDs (e.g., cameras) need to run DNNs for real-time video analytics. However, these devices are often computation-constrained (e.g., MCUs and tiny NPUs) with extremely limited memory (e.g., KB-level SRAM, and MB-level storage). On the one hand, it is difficult to execute large DNNs on these resource-constrained devices. On the other hand, streaming video data to cloud (for inference) can result in prohibitively high end-to-end latency and severe privacy disclosure [11]. In this sense, how to achieve efficient DNN inference (in terms of energy consumption and end-to-end latency) on local SDs becomes a key challenge.*

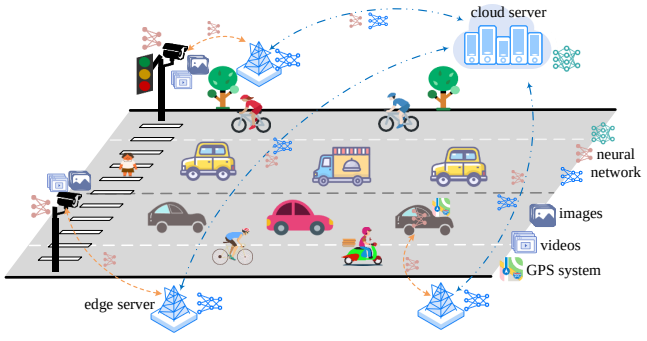


Fig. 1. A scenario of ML-based EC application in the smart city.

A natural way to solve this issue is cloud-edge-end collaborative inference and fine-tuning [12], [13], where a series of base DNN models can be selected, quantized, and partitioned online in cloud servers (CSs), and then the deep layers can be offloaded onto suitable ESs with different resource constraints for collaborative inference, where the shallow layers are offloaded to SDs for fine-tuning [14], [15], [31]. However, it is not trivial to implement such cooperation due to the complex mutual adaptation of the decisions concerning:

- **Model selection**. Although a series of well-trained base models are available on the cloud server (CS), the discrepancy between the private dataset on the SD and the

Yue-e Zhou and Lianbo Ma are with College of Software, Northeastern University, Shenyang 110819, China. E-mail: yueezhou1212@gmail.com, malb@swc.neu.edu.cn.

Xingwei Wang is with School of Computer Science and Engineering, Northeastern University, Shenyang, 110819, China. E-mail: wangxw@mail.neu.edu.cn.

Qing Li is with the Peng Cheng Laboratory, Shenzhen, Guangdong 518055, China. E-mail: andyliqing@gmail.com.

Carla Fabiana Chiasserini is with Politecnico di Torino, 10129 Torino, Italy, also with CNIT, 43124 Parma, Italy, and also with IEIIT-CNR, 10129 Torino, Italy. E-mail: carla.chiasserini@polito.it.

Guangjie Han is with the Department of Information Science and Engineering, Hohai University, Changzhou, 213022, China. E-mail: hanguangjie@gmail.com.

*:corresponding author

CS dataset results in these base models being incompatible with the SD. Consequently, we need to fine-tune the base model to enhance their adaptation to the local dataset. A critical question arises: How to identify the most suitable DNN model for fine-tuning when considering the variability among different SDs and their respective datasets [16], [17]?

- **Compression selection**. Given the limitation of SDs' resources, a discrepancy arises between the model size and SD's computational capacity. Consequently, the DNN models have to be compressed to accommodate the limited local resources. In this study, we chose model quantization as target compression solution, which can transform the full-precision model into a low bit-width representation to reduce the model size. However, how to quantize DNN models into appropriate bit-widths while preserving its performance with only a minimal decrement is a key question [18], [19].
- **Partitioning selection**. For resource-constrained SDs, it is essential to leverage the computational resources (not data) of ESs to assist in performing inference and fine-tuning tasks, while strictly preserving the data privacy of SDs. To this end, the DNN model is partitioned into shallow and deep layers, where the shallow layer is executed on SDs, and the deep layer is offloaded to ESs. Importantly, only intermediate feature representations generated by SDs are transmitted to ESs for processing, without involving raw data. How to well partition the model according to the resource availability between ESs and SDs is an important issue [20], [21].
- **Edge server selection**. During the fine-tuning process, ESs play a crucial role in receiving, processing, and transmitting diverse models from both CSs and SDs. We need to consider the computational and communication resources of each ES, as well as the compatibility of these resources with the SDs. Based on this consideration, it is essential to select optimal ESs to guarantee the efficiency and stability of the fine-tuning process [22], [23].

There are two main reasons why such *mutual adaptation* is very difficult:

- **Resource complexity.** The computational and communication resources are unevenly distributed across different edge servers, whose availability may dynamically vary in both spatial and temporal dimensions. Decisions made without considering changes in resources may be invalid.
- **Mutual coupling.** These decisions are tightly coupled with each other and need to be considered and formulated simultaneously. Any small slight move in one decision may affect the situation as the whole system.

As detailed in Section VII, existing studies [24]–[26] aim to address one or another of the aforementioned decision problems, rather than joint optimization for all these coupled decisions. To fill this gap, we in this paper propose a joint-decision framework called **J**oint **Q**uantization and **O**ffloading for **D**NN **I**nference (**JQODI**), which considers all the required decision problems through cloud-edge-end collaboration. In JQODI , all these decisions are formulated as a multi-dimensional optimization problem, aiming to minimize the overall energy consumption of the learning-inference process by jointly optimizing decision choices in terms of base model, compression, partition, and offloading strategy.

Especially, JQODI works well for inference acceleration of DNNs in situations where (i) the terminal datasets cannot be shared, (ii) different DNN models are available to choose from, and (iii) the inference is achieved via edge-end collaboration. A major novelty of JQODI lies in the ability to utilize different DNN models (e.g., full-precision model, deep layers model, and shallow layers model) across different stages of the cloud-edge-end collaboration (as shown in Figure 4 in Section III), by switching among them as needed (e.g., quantizing, and partitioning). For each stage, JQODI determines optimal decision for specific operation (e.g., quantization bit-width, and partitioning point), and selects the most appropriate resources (from ESs) for each model. Currently, the benefits derived from model quantization and partition must be balanced against the associated costs (i.e., time and energy), which entail the allocation of additional resources.

Our main contributions can be summarized as follows:

(1) *Joint-decision model formulation*: We construct a multi-dimensional optimization model for inference acceleration of large DNN models in the cloud-edge-end collaboration environments. Our model aims to jointly optimize all the real-time and concurrent decisions regarding: (i) the selection of DNN models, involving both full-precision and quantization versions; (ii) the determination of the partitioning point for quantized models; and (iii) the offloading strategy, i.e., the allocation of ESs, for edge-end inference. The optimization objective is to minimize the energy consumption, which accounts for both fine-tuning and inference costs, while ensuring learning time and accuracy. Unlike existing studies that focus on a single aspect at a time while treating other factors as fixed, JQODI addresses scenarios where all these decision dimensions can be controlled, thereby offering better flexibility and higher-quality decisions.

(2) *Algorithm strategies*: We propose JQODI algorithm to solve above joint-decision problem, which is essentially a large-scale combinatorial problem, involving massive possible solutions to choose from. First, we utilize an approximate dynamic programming (ADP) strategy to explore the most feasible solutions, at the same time use a time-energy tree diagram to represent the learning process. Second, since the joint decision process is mutual coupling, which indicates the model switching decisions may have various unexpected effects, we combine theoretical insights on DNN convergence with predictive analytics to estimate the effect of potential decisions. In this way, our JQODI achieves near-optimal decisions with a polynomial complexity.

(3) *Performance evaluation*: We evaluate the performance of JQODI in real-world cloud-edge-end scenarios for DNN collaborative inference acceleration. JQODI can generate training/inference strategy suitable for specific end device, achieving target learning quality and latency constraint at low energy consumption. In fact, JQODI's decisions are always very close and even the same to the optimal decision, significantly better than those made by state-of-the-art methods.

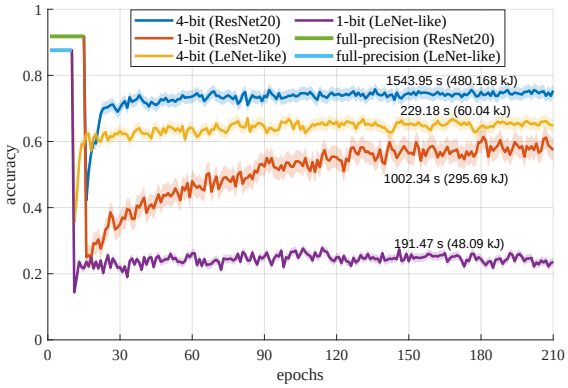


Fig. 2. Accuracy vs. training epochs for different models and quantization bit-width. Straight lines — and — represent the accuracy of full precision models, while polylines —, —, —, and — denote the accuracy of quantization models. The numbers in the figure represent the *time taken (energy consumption)* of different schemes, where the time is measured in seconds and the energy is measured in kilojoules. At the beginning of quantization, a sudden drop in accuracy occurs.

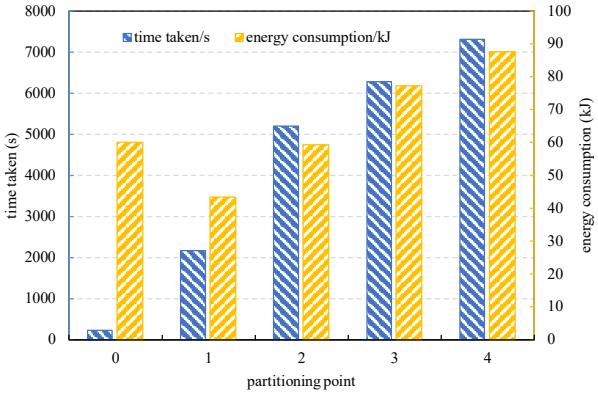


Fig. 3. Time taken vs. energy consumption for different partitioning points, where partition point 0 represents the overall running of the model on the NVIDIA GeForce RTX 3090 (ES). The model used in this figure is LeNet-like, with a 4-bit quantization, and 200 epochs running. The blue bar and left axis represent time consumption in seconds, while the yellow bar and right axis represent energy consumption in kilojoules.

II. A Motivating Example: The Importance of Joint Decision-Making

We now seek to illustrate the importance of joint decision-making in the collaborative training/inference process that integrates model switching among model quantization, partition and offloading, and also emphasize the benefits and challenges in formulating and optimizing joint decision. For better understanding, we adopt a real hardware testbed for collaborative inference (as shown in Figure 8 in Section VI). As a study case, we employ a Raspberry Pi5 and a NVIDIA A100-SXM4-80GB on behalf of SDs and CS, respectively. Besides, a NVIDIA GeForce RTX 3090 and a NVIDIA GeForce GTX 1080 are used to simulate ESs with different specifications. In particular, we evaluate the following cases:

- The nodes (e.g., CSs, ESs, and SDs) perform an image classification task using the ResNet-20[1] [27] or the LeNet-like[2] [28] models;
- The cloud dataset consists of all images from CIFAR-10 [29], while the SD dataset consists of 2000 training images and 500 validation images, which are randomly selected from 10 classes in CIFAR-100 [29].
- The candidate quantization bit-widths are 4-bit and 1-bit, which are corresponding to the compression rates of 87.5% and 96.875%, respectively.

Four decisions should be made: (i) the model to perform classification tasks, (ii) the quantization bit-width to compress the model, (iii) the partitioning point to divide the model, and (iv) the ES to perform deep layer. Note that the model accuracy depends only on the first two decisions, while the last two decisions mainly affect energy consumption and latency. Figures 2-3 show the effects of such decisions, which lead to the following observations.

[1] $16C3 - Block(16,3,1) - Block(32,3,1) - Block(64,3,1) - FC$, where $C3$ is a 3×3 ReLU convolution layer, $Block$ is a residual block containing three convolutional layers, and FC is a fully-connected layer.

[2] $64C3-MP2-128C3-MP2-128C3-MP2-1024FC-10SVM$, where $MP2$ is a 2×2 max-pooling layer, and SVM is a $L2-SVM$ output layer using the cross entropy loss function.

Observation 1: Small models (e.g., LeNet-like) with small quantization bit-widths (e.g., 1-bit) consume less energy and time, but result in lower model accuracy, which demonstrates the necessity of selecting optimal base models as well as quantization bit-widths for trade-off between efficiency and accuracy.

Observation 2: Not all choices are feasible, e.g., when compressing the LeNet-like model with 1-bit quantization, the model accuracy drops sharply and cannot be restored through fine-tuning, which indicates the requirement of identifying the feasibility of candidate decision.

Observation 3: SD consumes less energy but requires a longer time, while ES consumes more energy but with faster speed. This shows the importance of edge-end collaboration in dealing with time and energy constrains.

Observation 4: With the increase of the partitioning point ID, the overall energy consumption shows a trend of first decreasing and then increasing, rather than always increasing (or decreasing), which emphasizes the necessity of searching optimal partitioning point.

To sum up, the collaborative training/inference driven by model quantization and partition has significant benefits in terms of time and energy consumption; however, its effects are hard to capture and foresee. These benefits depend upon the decisions for the chosen base models, quantization bit-widths, partitioning points, and offloading strategy. Thus, it is necessary to jointly make all the decisions, accounting for their interactions through a comprehensive system model.

TABLE I
KEY NOTATIONS IN THIS PAPER

Notation	Description
M_a	the a-th DNN model version in the cloud server
π_q	the candidate quantization bit width
$\mathcal{L}_a$	the loss value of the model M_a
F_a	the computation burden of the model M_a
v_c	the data processing speed of CS
ε_c	the energy consumed of CS processing one unit of data
v_x	the data processing speed of edge server ES_x
ε_x	the energy consumption of ES_x processing one unit of data
v_y	the data processing speed of end device SD_y
ε_y	the energy consumption of SD_y processing one unit of data
$v_{(c,x)}$	the data transmission speed between ES_x and CS
$\varepsilon_{(c,x)}$	the energy consumption for transmitting one unit of data between ES_x and CS
$b_{(x,y)}$	the available bandwidth between ES_x and SD_y
$v_{(x,y)}$	the data transmission speed between ES_x and SD_y
$\varepsilon_{(x,y)}$	the energy consumption for transmitting one unit of data between ES_x and SD_y
$M_{(a,q)}$	the π_q-bit quantization model of M_a
$d^i_{(a,q)}$	the computational burden of the i-th layer/block of $M_{(a,q)}$
$g^i_{(a,q)}$	the size of the i-th layer/block of $M_{(a,q)}$
$\mathcal{T}(k)$	the end time of epoch k
$\mathcal{E}(k)$	the cumulative energy consumption of the first k epochs
$\mathcal{L}(k)$	the loss function on the test dataset of the SD, computed at epoch k being over
$t(k)$, $e(k)$	the time taken and energy consumption of epoch k
$\ell(k)$	the variation in the loss function value of the k-th epoch
$\mathcal{T}_{max}$	the maximum time limit for the training process
$\mathcal{L}_{max}$	the target accuracy required for the training process
τ^{run}	the time consumption of executing computing tasks
ε^{run}	the energy consumption of executing computing tasks
τ^{tran}	the time consumption of executing transmission tasks
ε^{tran}	the energy consumption of executing transmission tasks
$\tau_{(a,q)}$	the time taken to quantize M_a as $M_{(a,q)}$
$\varepsilon_{(a,q)}$	the energy consumption to quantize M_a as $M_{(a,q)}$
$\gamma_{(a,q)}$	the loss changes caused by compressing the full-precision model M_a as the quantization $M_{(a,q)}$
γ^{update}	the contribution of updating the quantization model $M_{(a,q)}$ of the k-th epoch with the full-precision model of the $k-1$-th epoch
γ^{run}	the effect of fine-tuning an epoch for model $M_{(a,q)}$
$K_{(a,q)}$	the number of epochs required to train model $M_{(a,q)}$
Θ	a virtual vertex that identifies all feasible paths

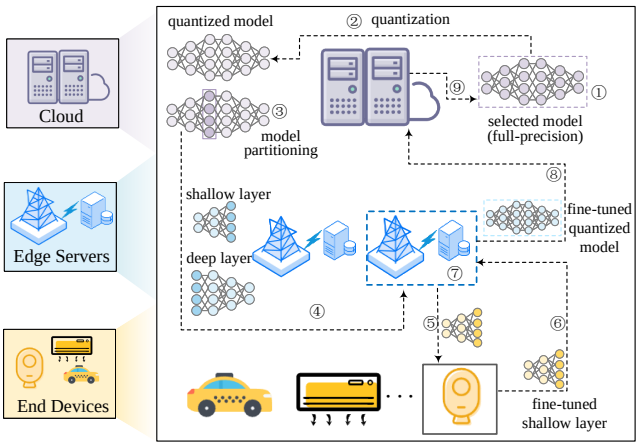
III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we first introduce the system model, followed by the problem formulation. Following [4], without loss of generality, we have simplified the relevant settings. The key notations used in this paper are listed in Table I.

A. System Overview

As shown in Figure 4, we consider a cloud-edge-end collaboration system for quantization, partitioning, and fine-tuning of DNN models, which consists of a CS, a set of ESs, and a set of heterogeneous SDs. The key components of the system are shown as follows:

1) Cloud Server (CS): The goal of CS is to accomplish a sequence of decision-making tasks by cooperating with ESs and SDs in the system: First, CS makes a decision of selecting a suitable basic model M_a from a set of DNN version candidates (i.e., $\mathcal{M} = \{M_1, \cdots, M_A\}$, where $M_a = \{\mathcal{L}_a, F_a\}$



① select a DNN model from basic models on the CS
② quantize the selected DNN model
③ determine the partitioning point for dividing the model
④ select an ES and transmit the partitioned deep and shallow layers to ES
⑤ ES transmits the shallow layer to the ED
⑥ ED transmits the fine-tuned shallow layer to ES
⑦ ES accepts shallow layer and continues to perform deep layer, ultimately generating a complete quantization model
⑧ ES transmits the fine-tuned quantization model to CS
⑨ CS uses the quantization models to update the full-precision model
note: steps 1-4 are performed simultaneously

Fig. 4. The proposed system model of cooperative training process.

denotes the attributes in the a-th DNN version, $\mathcal{L}_a$ and F_a denote the loss value and computational burden of the a-th version of M_a, respectively), which are fully pre-trained in cloud platform; Second, it needs to decide a specific bit-width π_q from candidate set (i.e. $\Omega = \{\pi_1, \cdots, \pi_Q\}$) to quantize the target M_a; Next, it calculates the optimal partitioning points to divide the quantized M_a into several sub-models; Then, it offloads the above sub-models onto appropriate ESs for collaborative inference. In this process, we fine-tune M_a through cloud-edge-end collaboration to let it adapt to the local dataset of the corresponding SD. The data processing speed of CS is v_c and the energy consumption to process one unit of data is ε_c. The time taken and energy consumed by π_q-bit quantization are $\tau_{(a,q)}$ and $\varepsilon_{(a,q)}$, respectively.

2) Edge Servers (ESs): For the set of ESs $\mathcal{ES}$=$\{ES_1, \cdots, ES_x, \cdots, ES_X\}$ in the EC system, each ES is connected to a base station through high-speed cables (such as fiber optic), offering communication resources. For simplicity, in this work, ES is considered to be equivalent to BS. For ES_x, its data processing speed is v_x and the energy consumption for processing one unit of data is ε_x. Both ES and CS communicate through internal IP addresses, and the data transmission speed between ES_x and CS is $v_{(c,x)}$, with each unit of data transmission consuming energy of $\varepsilon_{(c,x)}$.

3) End Devices (SDs): In the system, the set of SDs $\mathcal{SD}$=$\{SD_1, \cdots, SD_Y\}$ require a DNN model to process their own local privacy data. For each SD (SD_y), its data processing speed is denoted as v_y and the energy consumption for processing one unit of data is ε_y. Due SDs' resources, both CS and ES are required to assist SDs for fine-tuning DNN. ES_x and MD_y utilize wireless communication with the total available bandwidth $b_{(x,y)}$, and the energy consumption $\varepsilon_{(x,y)}$ for transmitting one unit of data. Besides, according to the Shannon's theorem [32], the data transmission speed between

ES_x and MD_y can be calculated as $v_{(x,y)} = b_{(x,y)} \cdot log_2(1+\sigma)$, where σ is the ratio of signal over noise. Similar to [22], for ease of modeling, we assume that σ is predictable and constant.

B. Computing and Communication Models

Deep layer and shallow layer. Upon selecting the DNN version $M_a \in \mathcal{M}$ and the quantization bit-width π_q, the resultant quantized model can be denoted as $M_{(a,q)}$. Let $d^i_{(a,q)}$ and $g^i_{(a,q)}$ denote the computational burden and the size of the i-th layer/block in the model $M_{(a,q)}$, respectively. For $M_{(a,q)}$ with n layers/blocks, since the partitioning point can be placed between any two layers/blocks, we can obtain the optional partition point set $\mathcal{SP} = \{p_1, p_2, \cdots, p_{n-1}\}$. Through the j-th partitioning point, $M_{(a,q)}$ can be divided into two parts: the DNN end layers/blocks (shallow layers/submodels) $\{l_1, l_2, \cdots, l_j\}$ executed on the SD and the DNN edge layers/blocks (deep layers/submodels) $\{l_{j+1}, l_{j+2}, \cdots, l_n\}$ executed on the ES.

Cloud-edge transmission. Prior to initiating the fine-tuning, the CS needs to transmit the deep sub-model and shallow sub-model derived from partitioned model to the ES for model offloading. Subsequently, the ES will transmit the shallow layer to the SD. The volume of data to be transmitted is denoted as:

$$D^{c\to e}_{(a,q)} = \sum_{i=1}^{n} g^i_{(a,q)}, \quad (cloud \to edge), \tag{1}$$

$$D^{e\to d}_{(a,q,j)} = \sum_{i=1}^{j} g^i_{(a,q)}. \quad (edge \to end). \tag{2}$$

End device fine-tuning. When receiving the shallow sub-model transmitted by CS, the corresponding SD needs to fine-tune it using the local dataset. Then, the local computational burden is expressed as:

$$D^{ed}_{(a,q,j)} = \sum_{i=1}^{j} d^i_{(a,q)}. \tag{3}$$

End-edge transmission. When the fine-tuning of SD is completed, the latest shallow layer needs to be transmitted to the ES for performing the deep layer. The amount of transmission data $D^{d\to e}_{(a,q,j)}$ $(end \to edge)$ is the same as Eq. (2).

Edge server fine-tuning. Upon receiving the shallow layer from SD, the ES continues to fine-tune the deep layer. The corresponding computational burden is defined as:

$$D^{es}_{(a,q,j)} = \sum_{i=j+1}^{n} d^i_{(a,q)}. \tag{4}$$

Edge-cloud transmission. When the fine-tuning of ES is completed, the latest quantization model $M_{(a,q)}$ is transmitted to CS for updating full-precision M_a. The volume of transmission data $D^{e\to c}_{(a,q,j)}$ $(edge \to cloud)$ is the same as Eq. (1).

C. Latency, Energy Consumption and Accuracy Modeling

Let k represent the current epoch, $\mathcal{T}(k)$ be the end time of epoch k, $\mathcal{E}(k)$ denote the cumulative energy consumption of the first k epochs, and $\mathcal{L}(k)$ be the loss function on the test dataset of the SD, computed at epoch k being over. Correspondingly, $t(k)$ and $e(k)$ represent the time taken and the energy consumption by epoch k, respectively, while $\ell(k)$ denotes the variation in the loss function value of the k-th epoch, which is usually a negative value.

Importantly, each epoch corresponds to one complete learning-inference run, which consists of (i) a fine-tuning phase to adapt the selected model to the local dataset, followed by (ii) an inference execution to evaluate the updated model and determine convergence and accuracy satisfaction.

Notice that the time and energy variations depend on the joint decision in terms of the selected M_a, the quantization bit-width (i.e., compression ratio) $M_{(a,q)}$, the partitioning decision $M_{(a,q,j)}$, and the adopted ES ES_x. Also, each of them includes two components, i.e.,

$$\begin{aligned} t(k) =& \mathcal{T}(k) - \mathcal{T}(k-1) \\ =& \tau^{run}(M_{(a,q,j)}, ES_x) + \tau^{tran}(M_{(a,q,j)}, ES_x), \end{aligned} \tag{5}$$

$$\begin{aligned} e(k) =& \mathcal{E}(k) - \mathcal{E}(k-1) \\ =& \varepsilon^{run}(M_{(a,q,j)}, ES_x) + \varepsilon^{tran}(M_{(a,q,j)}, ES_x), \end{aligned} \tag{6}$$

where τ^{run}/ε^{run} represents the time/energy consumption of executing computing tasks during fine-tuning over a set of cloud-edge-end nodes, and τ^{tran}/ε^{tran} denotes those of executing transmission tasks. Note that we omit the $M_{(a,q,j)}$ and ES_x for simplicity in the subsequent sections.

During the cloud-edge-end fine-tuning process, the time taken (τ^{run}) and the energy cost (ε^{run}) are consumed by the following calculations: the fine-tuning performed by SD, ES, and CS, and the quantization performed by CS. Consequently, the total time and energy consumed for calculations in one epoch are defined as:

$$\begin{aligned} \tau^{run} &= \tau_{ed} + \tau_{es} + \tau_{cs} + \tau_a, \\ &= \frac{D^{ed}_{(a,q,j)}}{v_y} + \frac{D^{es}_{(a,q,j)}}{v_x} + \frac{F_a}{v_c}, \end{aligned} \tag{7}$$

$$\begin{aligned} \varepsilon^{run} &= \varepsilon_{ed} + \varepsilon_{es} + \varepsilon_{cs} + \varepsilon_a \\ &= D^{ed}_{(a,q,j)} \cdot \varepsilon_y + D^{es}_{(a,q,j)} \cdot \varepsilon_x + F_a \cdot \varepsilon_c. \end{aligned} \tag{8}$$

Similarly, the calculation of τ^{tran} and ε^{tran} during the transmission process consists of four parts: i) deep layers and shallow layers (CS→ES), ii) shallow layers (ES→SD), iii) fine-tuned shallow layers (SD → ES), and iv) fine-tuned quantization model (ES→CS), i.e.,

$$\begin{aligned} \tau^{tran} &= \tau_{c\to e} + \tau_{e\to d} + \tau_{d\to e} + \tau_{e\to c} \\ &= \frac{D^{c\to e}_{(a,q,j)}}{v_{(c,x)}} + \frac{D^{e\to d}_{(a,q,j)}}{v_{(x,y)}} + \frac{D^{d\to e}_{(a,q,j)}}{v_{(x,y)}} + \frac{D^{e\to c}_{(a,q,j)}}{v_{(c,x)}} \\ &= 2 \cdot (\frac{D^{c\to e}_{(a,q,j)}}{v_{(c,x)}} + \frac{D^{e\to d}_{(a,q,j)}}{v_{(x,y)}}), \end{aligned} \tag{9}$$

$$\begin{aligned} \varepsilon^{tran} &= \varepsilon_{c\to e} + \varepsilon_{e\to d} + \varepsilon_{d\to e} + \varepsilon_{e\to c} \\ &= 2 \cdot (D^{c\to e}_{(a,q,j)} \cdot \varepsilon_{(c,x)} + D^{e\to d}_{(a,q,j)} \cdot \varepsilon_{(x,y)}). \end{aligned} \tag{10}$$

For some ineffective cases about the selection of partitioning points or ESs, we set τ^{run}, ε^{run}, τ^{tran}, and ε^{tran} to ∞.

The *evolution* of the loss function is formulated as:

$$\begin{aligned}\ell(k) =&\mathcal{L}(k) - \mathcal{L}(k-1)\\ =&\gamma^{update}(M_{(a,q)}, k-1, k) + \gamma^{run}(M_{(a,q)}, k).\end{aligned} \tag{11}$$

Again, Eq. (11) includes two components: $\gamma^{update}-$ the contribution of updating the quantized model $M_{(a,q)}$ of the k-th epoch with the full-precision model M_a of the $k-1$-th epoch, and $\gamma^{run}-$ the effect of fine-tuning an epoch for $M_{(a,q)}$. In general, γ^{run} and γ^{update} are less than zero (i.e., the loss decreases).

The sum of these components leads to the difference between the loss at the current epoch k and that of epoch $k-1$, i,e., the result of the action enacted at k. Assuming that the total number of epochs that $M_{(a,q)}$ needs to be fine-tuned is $K_{(a,q)}$, we can formulate the final loss of $M_{(a,q)}$ as:

$$\mathcal{L} = \mathcal{L}_a + \gamma_{(a,q)} + \sum_{k=1}^{K_{(a,q)}} \ell(k), \tag{12}$$

where $\gamma_{(a,q)}$ corresponds to the loss changes caused by compressing the model M_a into $M_{(a,q)}$ for the first time. Accordingly, $\gamma_{(a,q)} > 0$, as compressing model will increase the loss value. Impossible models and quantization bit-widths are associated with $\gamma_{(a,q)} = \infty$.

D. Problem Definition

Considering the critical requirement of sustainable ML, our objective is to minimize the total energy used for fine-tuning, while ensuring that the loss decreases to a predefined threshold $\mathcal{L}_{max}$ within a specified duration $\mathcal{T}_{max}$:

$$\begin{aligned}&\min\ \ \mathcal{E} = \varepsilon_{(a,q)} + \sum_{i=1}^{K_{(a,q)}} e(i),\\ s.t.&\\ &\mathcal{T} = \sum_{i=1}^{K_{(a,q)}} t(i) < \mathcal{T}_{max},\\ &\mathcal{L} = \mathcal{L}_a + \gamma_{(a,q)} + \sum_{k=1}^{K_{(a,q)}} \ell(k) < \mathcal{L}_{max}.\end{aligned} \tag{13}$$

Note that, unlike existing studies, our goal is to minimize training energy without compromising training efficiency or learning accuracy, rather than solely minimizing latency or maximizing accuracy, we consider a joint optimization over energy, accuracy, and latency. During fine-tuning, unlike τ^{run}, τ^{tran}, ε^{run} and ε^{tran} (that can be calculated in advance), the $\gamma_{(a,q)}$, γ^{run}, γ^{update} and $K_{(a,q)}$ can only be estimated or predicted by CS responsible for orchestrating task allocation and model customization through estimators $\hat{\gamma}_{(a,q)}$, $\hat{\gamma}^{run}$, $\hat{\gamma}^{update}$ and $\hat{K}_{(a,q)}$, respectively. This reveals the fact that it is difficult to comprehend how to train a particular model on specific SDs for enhanced learning performance. In fact, all existing methods can only offer approximations and/or bounds to such quantities. In the following, we consider these estimators as provided; then, in Section I of the Appendix, we give a potential method that CS can employ to calculate them.

In fact, the task of estimating and modeling the learning performance of DNNs is independent and unrelated to our work; the primary goal of JQODI is to utilize insights into DNN performance—no matter the source—to make informed, good learning choices efficiently. Due to the discrete-time and combinatorial characteristics of the target problem, we propose an *approximate dynamic programming* (ADP) method (detailed below) to tackle complex combinatorial problems where the state of the system changes over time and the same decision-making process must be repeated across various periods.

E. ADP Formulation

First, we define the state space $\mathcal{S}$, the set of actions $\mathcal{A}$, and the cost function $\mathcal{C}$. The epoch k is formulated as $s(k) = (k, \ell(k), t(k), ES_x, j)$, the set of actions available from state $s(k)$ is formulated by all possible actions $a(k) = (ES_x, j) \in (\mathcal{ES}, \mathcal{SP})$. The cost function $\mathcal{C}(s(k), a(k))$ refers to the cost of executing action a while in state s at epoch k, as the corresponding consumed energy $\mathcal{C}(s(k), a(k)) = e(k)$. Such a cost comes directly from Eq. (6), i.e., $\mathcal{C}(s(k), a(k)) = \varepsilon^{run}(M_{(a,q,j)}, ES_x) + \varepsilon^{tran}(M_{(a,q,j)}, ES_x)$.

The desirable of being in state $s(k)$ is called as *value function*, denoted as $V(s(k))$, which requires a more complex and domain-specific definition. When the time is greater than $\mathcal{T}_{max}$ (i.e., $\mathcal{T}(k) > \mathcal{T}_{max}$) and the loss function exceeds $\mathcal{L}_{max}$ (i.e., $\mathcal{L}(k) > \mathcal{L}_{max}$), we assign the minimum value 0 to the value function $V(s(k))$ of state $s(k)$. When the time is less than or equal to $\mathcal{T}_{max}$ (i.e., $\mathcal{T}(k) \leq \mathcal{T}_{max}$) and the loss function is lower than $\mathcal{L}_{max}$ (i.e., $\mathcal{L}(k) < \mathcal{L}_{max}$), we assign the maximum value 1 to the value function $V(s(k))$ of state $s(k)$. For all other states, we compare the current loss $\mathcal{L}(k)$ and time $\mathcal{T}(k)$ with an ideal loss-time curve $\mathcal{L}^{ideal}(\mathcal{T})$ that: (i) starts at $\mathcal{L}(0)$ for $\mathcal{T} = 0$; (ii) ends at $\mathcal{L}_{max}$ for $\mathcal{T} = \mathcal{T}_{max}$, and (iii) follows a power law in between. This latter aspect is derived from findings consistently reported in both theoretical [33], [34] and experimental [35] studies. Consequently, the value of being in state $s(k)$ can be formulated as the difference between the ideal and the actual loss values, i.e.,

$$V(s(k)) = \text{logistic}\left(\mathcal{L}^{ideal}(\mathcal{T}(k)) - \mathcal{L}(k)\right), \tag{14}$$

where the value is normalized using a logistic function.

Theoretically, the dynamic programming (DP) problems can be solved via the optimization of Bellman's equation, which entails selecting, at each time step, the action that results in the lowest cumulative energy cost while satisfying the conditions that (i) the specified quality criterion is met, indicated by the state value at the final time step K being 1, and (ii) the action is executed prior to the expiration of the deadline $\mathcal{T}_{max}$.

$$\begin{aligned}&\min\nolimits_{a(k)\in\mathcal{A}} \sum_k \mathcal{C}(s(k), a(k)),\\ &s.t.\ V(s(K)) = 1,\ \mathcal{T}(K) < \mathcal{T}_{max}.\end{aligned} \tag{15}$$

When solving realistic DP problems, we confront two major challenges. First, the CS has no enough ability to foresee

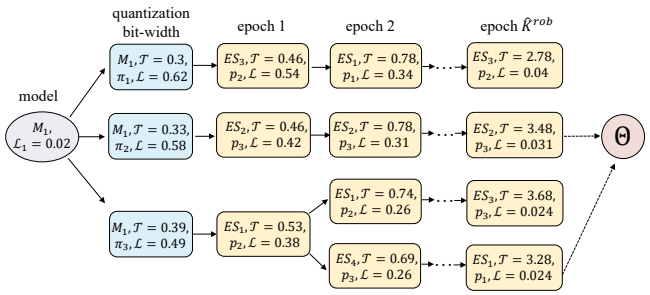


Fig. 5. Example of the JQODI time-energy tree diagram.

the future changes in the loss value $\ell(k)$ as well as the required total number of epochs K. Though deep reinforcement learning (DRL) has potential to deal with the above task, it requires a large amount of data for agent training, and suffers from limited generalization, which is not suitable to our target resource-limited EC scenarios. For this issue, we design an alternative strategy, i.e., an ADP framework, via utilizing low-complexity neural network estimators to predict the potential loss trajectories and the required total number of epochs K. Second, given the vast number of possible actions, it is difficult for the CS to selectively evaluate a subset of these actions at each iteration. These challenges are addressed in Sections IV and V, respectively.

IV. The JQODI Algorithm

The goal of JQODI is to enable CS to efficiently identify high-quality solutions for the problem presented in Eq. (15), which, as demonstrated subsequently, is NP-hard. The process of JQODI includes three main stages:

1) Construct a *time-energy tree diagram* that depicts potential decisions and their respective outcomes.
2) Utilize this graph to determine a feasible set of decisions based on the predicted loss trajectory.
3) Integrate time and energy-related data to select the optimal feasible solution for implementation.

Step 1: Time-energy tree diagram. The time-energy tree diagram is a directed graph constructed according to the following rules:

- There are three types of $vertices$ in the graph, with red and orange vertices indicating the selected model v_m and quantization bit-width v_q, respectively. The blue vertices depict the states of the system v_e and are marked with details such as the current edge server ES, the partitioning point p, and the total elapsed time $\mathcal{T}(k)$. To identify feasible solutions, we use the robust estimator $\hat{K}^{rob}_{(a,q)}$ to construct the graph.
- A directed edge is drawn between two vertices whenever an action making the system move from one state to another; each such edge is labeled with the energy expenditure of the related action, as in Eq. 6.
- Each vertex representing a feasible state of the system (i.e., $\mathcal{T}(\hat{K}^{rob}_{(a,q)}) < \mathcal{T}_{max}$) is further connected to a virtual vertex Θ.

The diagram is generated using the CreateTree function, as shown in Algorithm 1. First, it forms all vertices, embodying

Algorithm 1 Creating the time-energy tree diagram

Input: the set $\mathcal{M}$, $\mathcal{ES}$, Ω, and $\mathcal{SP}$;
Output: the time-energy tree diagram $(\mathbb{V}, \mathbb{E})$;
function CreateTree
1: $\mathbb{V} \leftarrow \{\Theta\}$, $v_m \leftarrow \emptyset$, $v_q \leftarrow \emptyset$, $v_e \leftarrow \emptyset$; ▷ set of vertices
2: **for** $M_a \in \mathcal{M}$ **do**
3: $v_m \leftarrow v_m \cup \{M_a\}$;
4: **for** $q_i \in \Omega$ **do**
5: $v_q \leftarrow v_q \cup (M_a, q_i)$;
6: **for** $k \in \{1, \cdots, \hat{K}^{rob}_{(a,q)}\}$ **do**
7: $v_e \leftarrow v_e \cup (ES_x, q_i, \mathcal{T}(k))$;
8: $\mathbb{V} \leftarrow \mathbb{V} \cup v_m \cup v_q \cup v_e$;
9: $\mathbb{E} \leftarrow \emptyset$; ▷ set of edges
10: **for** $v \in v_m$ **do**
11: **for** $v' \in v_q$ **do**
12: **if** $\tau_{(a,i)} \leq \mathcal{T}_{max}$ **then**
13: $E = \varepsilon_{(a,i)}$, $\mathcal{L} \leftarrow \mathcal{L}_a + \hat{\gamma}^{rob}_{(a,q)}$;
14: $v' \leftarrow (M_a, q_i, \tau_{(a,i)}, \mathcal{L})$;
15: $\mathbb{E} \leftarrow \mathbb{E} \cup (v, v', \text{weight} = E)$;
16: **else if** $\tau_{(a,i)} > \mathcal{T}_{max}$ **then**
17: $E = \infty$; ▷ infeasible
18: **for** $v \in v_q$ **do**
19: **for** $v' \in v_e(k = 1)$ **do**
20: $\mathcal{T}' \leftarrow \mathcal{T} + \tau^{run}(v') + \tau^{tran}(v')$;
21: **if** $\mathcal{T}' \leq \mathcal{T}_{max}$ **then**
22: $E = \varepsilon^{run}(v') + \varepsilon^{tran}(v')$;
23: $\mathcal{L} \leftarrow \mathcal{L} + \hat{\gamma}^{update}_{rob}(v') + \hat{\gamma}^{run}_{rob}(v')$;
24: $v' \leftarrow (ES_x, q_i, \mathcal{T}', \mathcal{L})$;
25: $\mathbb{E} \cup (v, v', \text{weight} = E)$;
26: **else if** $\mathcal{T}' > \mathcal{T}_{max}$ **then**
27: $E = \infty$; ▷ infeasible
28: **for** $v \in v_e(k \leq \hat{K}^{rob})$ **do**
29: **for** $v' \in v_e((k+1) \leq \hat{K}^{rob})$ **do**
30: $\mathcal{T}' \leftarrow \mathcal{T} + \tau^{run}(v') + \tau^{tran}(v')$;
31: **if** $\mathcal{T}' \leq \mathcal{T}_{max}$ **then**
32: $E = \varepsilon^{run}(v') + \varepsilon^{tran}(v')$,
33: $\mathcal{L} \leftarrow \mathcal{L} + \hat{\gamma}^{update}_{rob}(v') + \hat{\gamma}^{run}_{rob}(v')$;
34: $v' \leftarrow (ES_x, q_i, \mathcal{T}', \mathcal{L})$;
35: $\mathbb{E} \cup (v, v', \text{weight} = E)$;
36: **else if** $\mathcal{T}' > \mathcal{T}_{max}$ **then**
37: $E = \infty$; ▷ infeasible
38: **if** $\mathcal{T} < \mathcal{T}_{max} \cap \mathcal{L} < \mathcal{L}_{max}$ **then**
39: $\mathbb{E} \leftarrow \mathbb{E} \cup \{v, \Omega\}$;
40: **return** $(\mathbb{V}, \mathbb{E})$;

all valid combination of model, quantization bit-width, ES, epoch, partitioning point, and taken time (Lines 1-8).

For each model M_a ($v \in v_m$), if the time taken for q_i-bit quantization is less than $\mathcal{T}_{max}$, an effective edge with a weight of $\varepsilon_{(a,i)}$ is added between vertices M_a and $(M_a, q, \tau_{(a,i)}, \mathcal{L})$, where $\mathcal{L}$ is calculated using the robust estimator (Lines 10-17). For each vertex $v \in v_q$ or v_e, determining the impact of executing action v' from v_e involves calculating the resultant elapsed time and the required energy (Lines 18-27 and Lines 28-37). If either of these values is ∞, executing action v' from state v is deemed infeasible, prompting us to consider the next action. Otherwise, an effective edge with a weight of E (Line 25 and Line 35) is added between vertices v and v' (Line 24 and Line 34), where $\mathcal{L}$ is calculated using the robust estimator (Line 23 and Line 33). Finally, if v is feasible, v is connected to Θ (Lines 38-39).

Figure 5 presents an example of time-energy tree diagram,

where the learning target is $\mathcal{L}_{max} = 0.035$ and the time limit is $\mathcal{T}_{max} = 3.5$. The initial point is model M_1 with a loss value of $\mathcal{L}_1 = 0.02$ (denoted by purple vertex). This model can be further processed with three potential quantization bit-widths: π_1, π_2, and π_3 (depicted as blue vertices). Throughout the fine-tuning phase, different configurations of ESs and partitioning points can be chosen for each training iteration (yellow vertices). Among these different combinations of the loss and execution time, only a few subsets are feasible, and then connected to decision node Θ (red vertex).

Step 2: Find available paths. Subsequently, JQODI utilizes a multi-tree traversal algorithm to ascertain viable paths. To counteract the potential detrimental effects of inaccuracies in loss estimation, which could undermine the feasibility of these paths, a time-energy tree diagram is constructed leveraging robust estimators of loss variation. This ensures that all paths terminating at a feasible node possess a high probability of actual feasibility. Therefore, JQODI applies Algorithms 2-3 to search for paths that: (i) originate from the initial vertex $M_a(v \in v_m)$, and (ii) terminate in a feasible state, namely, a vertex linked to Θ. These identified paths are compiled into a set denoted as $\mathcal{P}$ and are assigned weights corresponding to the sum of weights (i.e., energy consumption) of their edges.

Algorithm 2 Finding available paths

Input: the time-energy tree diagram $(\mathbb{V}, \mathbb{E})$;
Output: the path set $\mathcal{P}$;
$\mathcal{P} \leftarrow \emptyset$;
1: **for** $v \in v_m$ **do**
2: **TRAVERSAL()**; ▷ Invoke Algorithm 3
3: **return** $\mathcal{P}$;

Algorithm 3 TRAVERSAL

$\mathcal{P} \leftarrow \emptyset$; ▷ set of paths
function TRAVERSAL()
1: **if** $v == \Theta$ **then**
2: $p = (path, w)$;
3: $\mathcal{P} \leftarrow \mathcal{P} \cup p$;
4: **return**;
5: **for** $edge = (v, v', weight) \in \mathbb{E}$ **do**
6: $v = v'$;
7: $path \leftarrow path + v$;
8: $w \leftarrow w + edge(weight)$;
9: **TRAVERSAL()**; ▷ recursion
10: $path \leftarrow path - v$;
11: $w \leftarrow w - edge(weight)$;

Step 3: Making the best decision. After identifying the available paths and associated actions, utilizing robust estimators for decision-making can be overly prudent, potentially leading to increased energy expenses. Therefore, JQODI considers two additional elements — **opportunity and risk factor** — during action selection. These factors, along with the path weight, are combined into a score, and the action with the lowest score is chosen for implementation.

For each path $p \in \mathcal{P}$, scores are computed in the SelectPath function in Algorithm 4. The opportunity factor, i.e., $opp \geq 1$, is determined by the ratio of (i) the expected time to (ii) the robust time associated with the path p (Lines 2-4). The

Algorithm 4 Selecting the best path

Input: the path set $\mathcal{P}$;
Output: the best path $p*$;
function SelectPath
1: **for** $p \in \mathcal{P}$ **do**
2: $\mathcal{T}_{rob}(p) = \mathcal{T}\left(\hat{K}^{rob}_{(a,q)}(p)\right)$;
3: $\mathcal{T}_{exp}(p) = \mathcal{T}\left(\hat{K}^{exp}_{(a,q)}(p)\right)$;
4: $opp \leftarrow \mathcal{T}_{exp}/\mathcal{T}_{rob}$;
5: $\mathcal{L}_{rob}(p) = \mathcal{L}_a(p) + \hat{\gamma}^{rob}_{(a,q)}(p) + \sum_{k=1}^{\hat{K}^{rob}_{(a,q)}(p)} \left(\hat{\gamma}^{update}_{rob}(k) + \hat{\gamma}^{run}_{rob}(k)\right)$;
6: $\mathcal{L}_{exp}(p) = \mathcal{L}_a(p) + \hat{\gamma}^{exp}_{(a,q)}(p) + \sum_{k=1}^{\hat{K}^{exp}_{(a,q)}(k)} \left(\hat{\gamma}^{update}_{exp}(k) + \hat{\gamma}^{run}_{exp}(p)\right)$;
7: $risk \leftarrow \mathcal{L}_{exp}(p)/\mathcal{L}_{rob}(p)$;
8: $scores[p] \leftarrow p[w] \cdot \frac{risk}{opp}$;
9: **return** $p* = argmin_{p \in \mathcal{P}}\ scores[p]$;

underlying principle is to make it easier to choose paths with a more accurate estimate for the expected number of epochs, since the more accurate the estimated epoch, the closer the simulation results of the fine-tuning process of the path are to the real results. As for $risk \geq 1$, it is computed as the ratio if (i) the sum of expected loss value to (ii) the sum of the robust loss value associated with the path p (Lines 5-7). The high-level objective of the risk factor is to steer away from selecting paths with large differences between the robust loss value and the expected loss value, since the larger the difference between them, the poorer the stability of the path.

The score of path p is computed in Line 13 as p's weight, divided by the opportunity factor, and multiplied by the risk factor. Then, the path with the minimum score is identified as the optimal path. Note that the path with the lowest weight represents the decision with the minimal energy cost, when the edge weights are associated with the energy consumption of the corresponding actions. To this end, this step outputs the path that leads to the lowest energy cost.

A. Problem and Algorithm Analysis

Property 1: The joint decision-making problem involving the selection of model, quantization bit-width, partitioning points, and ESs is NP-hard.

Proof. To prove **Property 1**, we need to give the following preparations.

Problem Transformation. According to the definitions in Section III, our target joint decision-making problem can be rephrased as

$$\max\ -\left(\varepsilon_{(a,q)} + \sum_{k=1}^{K} e(k) \cdot \eta_{(a,q,j,x)}\right), \qquad (16)$$

s.t.

$$\eta_{(a,q,j,x)} \leq 1, \quad \eta_{(a,q,j,x)} \in \{0,1\}, \tag{15.1}$$

$$\tau_{(a,q)} + \sum_{k=1}^{K} t(k) \cdot \eta_{(a,q,j,x)} < \mathcal{T}_{max}, \tag{15.2}$$

$$\mathcal{L}_a + \gamma_{(a,q)} + \sum_{k=1}^{K} \ell(k) \cdot \eta_{(a,q,j,x)} < \mathcal{L}_{max}, \tag{15.3}$$

where $\eta_{(a,q,j,x)}$ represents the current decision state selected by the system. If the currently used model is M_a, the quantization bit-width is π_q, the partitioning point is l_j, the selected edge server is ES_x, and the value of $\eta_{(a,q,j,x)}$ is 1, otherwise $\eta_{(a,q,j,x)} = 0$.

Multi-dimensional Cost Knapsack (MCK) Problem. The MCK problem involves n items and a knapsack with a load-bearing capacity of L and a volume of W. The weight and volume used to place the i-th item in the knapsack is l_i and w_i, respectively; and the obtained value is v_i. The goal of the MCK problem is to solve which items can be packed into the knapsack for maximization of the total value, i.e., $I = (l_1, \cdots, l_n, w_1, \cdots, w_n, v_1, \cdots, v_n, n, \eta_i, L, W)$:

$$\begin{aligned} &\max \sum_{i=1}^{n} v_i \cdot k_i, \\ &s.t. \quad \sum_{i=1}^{n} l_i \cdot \eta_i \leq L, \quad \sum_{i=1}^{n} w_i \cdot \eta_i \leq W, \end{aligned} \tag{17}$$

where $\eta_i \in \{0,1\}$. If the item i is packed into the backpack, $\eta_i = 1$; otherwise, $\eta_i = 0$. It is well known that this knapsack problem is an NP-hard problem [37], [38].

Through comparing Eq. 16 with Eq. 17, we can obtain that if neglecting some polynomial-time items, our joint decision-making problem of Eq. 16 is is equivalent to the MCK problem, which has been proven to be NP-hard. Specifically, we can map I to an instance of target joint decision-making problem $I' = (l_i = t(k), w_i = \ell(k), v_i = e(k), n = K, \eta_i = \eta_{(a,q,j,x)}, L = \mathcal{T}_{max} - \tau_{(a,q)}, W = \mathcal{L}_{max} - \mathcal{L} - \gamma_{(a,q)})$. It is evident that the mapping can be completed within the polynomial time. If an algorithm that can resolve the target joint decision-making problem I' exists, it would likewise address the associated MCK problem I, and the reverse is true as well. Consequently, the MCK problem can be views as a specific instance of the joint decision-making problem, which is also NP-hard.

Property 2: The time complexity of our proposed algorithm JQODI is $\mathcal{O}(max\{A*Q*\mathcal{K},\ |v_m|*|v_q|,\ |v_q|*|v_e|,\ |v_e|^2\})$.

Proof. The time complexity of JQODI strategy is determined by the cumulative complexities of Algorithms 1−4. In Algorithm 1, the first loop can be executed up to $A*Q*\mathcal{K}$ times, where $\mathcal{K} = \text{argmax}_{(a,q)} \hat{K}^{rob}_{(a,q)}$. The second loop can iterate up to $|v_m|*|v_q|$ times, the third loop can be executed up to $|v_q|*|v_e|$ times, and the fourth loop can be executed up to $|v_e|*(|v_e|-1)$ times. Algorithms 2−3 aim to find the available paths via traversing every node of the tree diagram constructed by Algorithm 1. Therefore, the complexity of Algorithms 2−3 will not exceed the total number of nodes $|v_m+v_q+v_e|$ in the graph. Algorithm 4 aims to find the best one among all feasible paths, through iterating over the set $\mathcal{P}$ of feasible paths with a complexity of $|\mathcal{P}|$. We can easily determine that the contribution of Algorithm 1 to the complexity of JQODI dominates, while the complexity of Algorithms 2-4 is much lower than that of Algorithm 1. Consequently, the complexity of JQODI is $\mathcal{O}(max\{A*Q*\mathcal{K},\ |v_m|*|v_q|,\ |v_q|*|v_e|,\ |v_e|^2\})$.

Importantly, **Property 2** concerns the worst-case time complexity of JQODI, which in practice has substantially lower complexity [39].

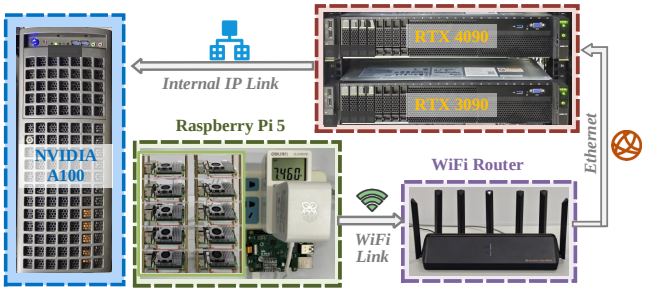


Fig. 6. The proposed system model of cooperative training process.

TABLE II
END DEVICES SPECIFICATIONS

Device	Hardware	Specifications	
Pi5	CPU	2.4GHz Quad Core Arm Cortex-A76	
	Memory	8GB LPDDR4X-4267MHz	
	GPU	No GPU	
	Power	Idle	3.75W
		Rated	12W

Rated power: the maximum power that can be continuously output under normal working conditions;

V. EXPERIMENTS AND RESULT ANALYSIS

In this section, we use a real hardware testbed to testify the performance of our proposed algorithm.

A. Cloud-Edge-End System Setup

Testbed Overview. As shown in Figure 6, we employ a set of Raspberry Pi5 as SD, running the Ubuntu 22.04 system to fine-tune shallow layer model and uploading intermediate features; a NVIDIA GeForce RTX 4090 and RTX 3090 are used to simulate ESs with different resource levels, running Ubuntu 20.04 system to fine-tune the deep model, transmit shallow model, and serve as a communication medium between CS and SD; a NVIDIA A100 GPU as a CS, running the Ubuntu 20.04 system for model storage and deployment of the JQODI algorithm for decision-making. SDs and ESs communicate wirelessly through Wi-Fi modules, while the connection between ES and CS is realized through internal IP addresses. The power consumption of CS and ES is monitored

TABLE III
EDGE SERVERS SPECIFICATIONS

Device	Hardware	Specifications	
RTX3090	CPU	12th Gen Intel(R) Core(TM) i9-12900	
	Memory	32GB	
	GPU	NVIDIA GeForce RTX 3090-24G	
	Power	Idle	95W
		Rated	350W
RTX4090	CPU	Intel(R) Xeon(R) Gold 6133 CPU@2.50GHz	
	Memory	64GB	
	GPU	NVIDIA GeForce RTX 4090-24G	
	Power	Idle	35W
		Rated	450W

TABLE IV
CLOUD SERVER SPECIFICATIONS

Device	Hardware	Specifications	
A100	CPU	Intel(R) Xeon(R) Gold 6133 CPU@2.50GHz	
	Memory	64GB	
	GPU	NVIDIA A100-SXM4-80GB	
	Power	Idle	66W
		Rated	400W

using 'nvidia-smi', while SD's (Raspberry Pi) energy usage is measured using a Deli DL333501C power monitor, as shown in Fig. 6. Tables II-IV present their specifications and reported power parameters.

Data Requests and DNN Architectures. The data requests from SDs are generated based on CIFAR-10 [29] and CIFAR-100 [29]: Data 1, simulating homogeneous datasets, has 2000 training images and 500 validation images from the test set of CIFAR-10; Data 2, simulating heterogeneous datasets, has 5000 training images and 1000 validation images from 10 classes in CIFAR-100. We consider three DNN architectures, including LeNet-like, VGG-like[3] [30], and ResNet20, all of which have been fully trained on CIFAR-10 in cloud server.

Algorithm Settings. We implement Algorithms 1-4 through quantization-aware training [31] based on Pytorch 2.4.1 and Python 3.8. For each model, the available quantization bit-widths follow the set {2-bit, 4-bit, 6-bit, 8-bit}. We empirically set four optional partitioning points for LeNet-like[4] and ResNet20[5], and five for VGG-like model[6]. Note that the energy consumption of idle nodes is not taken into account [4], [40], [41]. Then, for comparison, we set significantly different time constraints for the two experiments: the time limit ($\mathcal{T}_{max} = 180$ seconds) is set for Data 1, and the long time limit ($\mathcal{T}_{max} = 2000$ seconds) is set for Data 2.

Benchmark Solutions. The benchmark strategies for comparison include: (i) *Optimum*: the optimal decision achieving the minimum energy consumption via exhaustive brute-force search is only applicable to small-scale instances; therefore, it is used solely as a sanity-check upper bound; (ii) *Full Model*: the full-precision model without any quantization, which utilizes brute-force search to determine model version, partition point, and ES; (iii) *Minimal Model*: the 1-bit quantization model, where the optimal selection of model version, partition point, and ES is determined using brute-force search. Note that most state-of-the-art works [21], [42], [43] envision full-precision model, and thus their performance is represented by *Full Model*.

[3] $2 \times 128C3 - MP2 - 2 \times 256C3 - MP2 - 1024FC - 10SVM$

[4] The partitioning point of LeNet-like: $64C3 - MP2 - p_1 - 128C3 - MP2 - p_2 - 128C3 - MP2 - p_3 - 1024FC - p_4 - 10SVM$

[5] The partitioning point of ResNet20: $16C3 - p_1 - Block(16,3,1) - p_2 - Block(32,3,1) - p_3 - Block(64,3,1) - p_4 - FC$

[6] The partitioning point of VGG-like: $128C3 - p_1 - 128C3 - MP2 - p_2 - 256C3 - p_3 - 256C3 - p_4 - MP2 - 1024FC - p_5 - 10SVM$

TABLE V
ENERGY CONSUMPTION OF SYSTEM COMPONENTS AND JQODI OVERHEAD

Task	Total Energy	JQODI Energy
Data 1	7651.3J	28.35J
Data 2	12091J	36.75J

TABLE VI
PREDICTION OF THE TOTAL NUMBER OF EPOCHS RELATED TO DATA 1

$\mathcal{L}_{max}$	Model	$\mathcal{L}$	π_q	K	$\hat{K}^{exp}$	$\hat{K}^{rob}$
0.02	LeNet-like	0.0186	2-bit	126±6	126	132
			4-bit	44±4	44	48
			6-bit	33±3	33	36
			8-bit	26±3	26	29
	ResNet20	0.0156	2-bit	∞	-	-
			4-bit	56±4	56	60
			6-bit	48±2	48	50
			8-bit	42±2	42	44
	VGG-like	0.0188	2-bit	71±5	71	76
			4-bit	43±3	43	46
			6-bit	39±3	39	42
			8-bit	29±2	29	31

[1] $\mathcal{L}$: the loss value of the corresponding full-precision model
[2] $\mathcal{L}_{max}$: the target loss value of quantized model
[3] K: the number of epochs required for the quantized model to reach the target loss $\mathcal{L}_{max}$ during fine-tuning
[4] $\hat{K}^{exp}$ and $\hat{K}^{rob}$: the expected-value estimator and the robust-value estimator for K, respectively

JQODI Energy Consumption. Note that, JQODI algorithm is executed on cloud sever and is triggered only during the initial phase or when there is a significant changes in resource status. The optimization results are reused across multiple task rounds, with an energy cost of approximately 3‰ of the total energy of the cloud-edge-end system, as shown in Table V.

B. Implementation of Total Epoch Quantity Prediction.

As described in Section I of the Appendix, we use two DNNs and an LSTM to estimate $\gamma_{(a,q)}$, γ^{run}, γ^{update}, and, $K_{(a,q)}$ respectively. Considering the distinctions arising from various configurations, we conduct individual prediction for each choice, encompassing selected model and quantization bit-width, as well as perform separate predictions for the number of epochs required for each dataset.

The prediction quality results are shown in Tables VI–VII, where the epoch number predictions are generally accurate

TABLE VII
PREDICTION OF THE TOTAL NUMBER OF EPOCHS RELATED TO DATA 2

$\mathcal{L}_{max}$	Model	$\mathcal{L}$	π_q	K	$\hat{K}^{exp}$	$\hat{K}^{rob}$
0.02	LeNet-like	0.0186	2-bit	145±7	145	152
			4-bit	53±3	53	56
			6-bit	36±3	36	39
			8-bit	33±2	33	35
	ResNet20	0.0156	2-bit	∞	-	-
			4-bit	59±3	59	62
			6-bit	47±2	47	49
			8-bit	50±2	50	52
	VGG-like	0.0188	2-bit	113±6	113	119
			4-bit	73±4	73	77
			6-bit	69±3	69	72
			8-bit	58±2	58	60

in all cases. From these results, we can obtain following observations: First, reducing quantization bit-width can decrease computational and energy costs, but increase the number of epochs required for fine-tuning. Second, a higher quantization bit-width does not always guarantee faster convergence to target loss. For example, in the case of Data 2, an 8-bit ResNet20 network consistently requires more epochs to achieve the target loss than the 6-bit ResNet20 network. This shows that the most energy-efficient configuration does not necessarily result in the model with the smallest size or lowest quantization bit-width. Therefore, employing learning optimization strategies such as JQODI is crucial to balance model efficiency and fine-tuning costs.

C. JQODI 's Performance on Data 1

1) We first assess the effectiveness of our proposed JQODI on Data 1. As shown in Figure 7, the result generated by the *Minimal Model* strategy corresponds to the performance of the LeNet-like network when selecting the second partition point and deploying the deep model at RTX 3090 with a 1-bit quantization; the *Optimum* strategy is to select the first partition point in the LeNet-like network with an 8-bit quantization and deploy the model at RTX 3090; the difference between *JQODI* and *Optimum* is that the quantization scheme adopted for the LeNet-like network is 6-bit; the results generated by *Full Model* correspond to the performance of the full-precision VGG-like network when partitioning at the first partition point and deploying the deep model at RTX 4090. Due to the stability of the network environment during the experiment, the most accurate quantization bit-width and partition points can always be accurately adopted for each epoch, as well as the combination of quantization bit-width and partition point is relatively fixed.

Figure 7(a) illustrates the energy consumption associated with each strategy for different loss values. The *Minimal Model* emerges as the least effective approach, failing to achieve $\mathcal{L}_{max}$, even without accounting for energy consumption. By contrast, the *Full Model* shows performance improvement at the cost of increased training effort. However, its energy consumption to reach the target loss value significantly exceeds that of *JQODI* and *Optimum*. This disparity arises because training a full-precision model incurs higher energy costs per epoch than compressed model. Moreover, the full-precision model is trained on a CS with extensive training data, whereas SDs suffer from lack of sufficient training data.

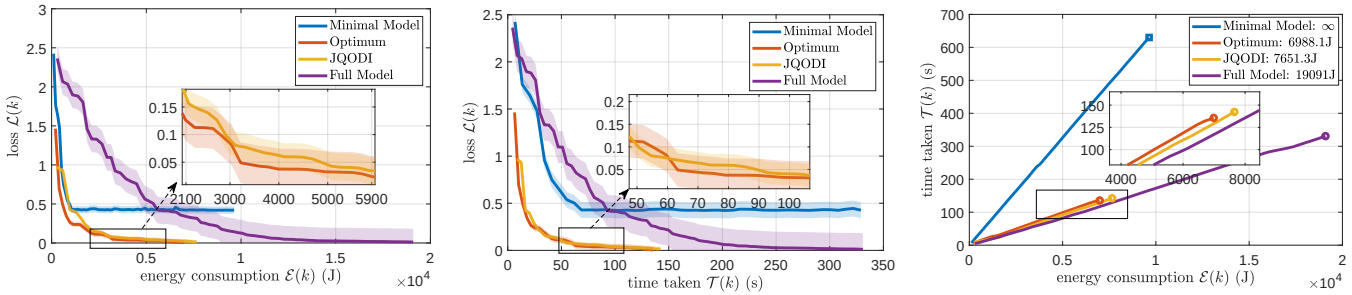
Figure 7(b) compares the training loss versus time taken curves of the benchmark strategy and *JQODI*. We observe that *JQODI* converges to the target loss $\mathcal{L}_{max}$ almost simultaneously with *Optimum*. More specifically, on Data 1, *JQODI* makes decisions that are very close to those of the optimal strategy, differing only slightly in quantization bit-width, i.e., 6-bit for *JQODI* compared to 8-bit for the optimal strategy. However, the ultimate objective of our optimization is energy consumption, with time consumption serving merely as a constraint. Therefore, Figure 7(c) highlights the energy consumption and time consumption of each strategy at the point of reaching the target loss.

As shown in Figure 7(c), although *Minimal Model* incurs significant time and energy costs, it consistently fails to achieve the target loss. While *Full Model* ultimately converges to the target loss, it entails considerable energy expenditure (19091 J) and time consumption (330.86 s) much longer than our acceptable range. Notably, our *JQODI* achieves performance that is only marginally inferior to the optimal solution in terms of both time and energy consumption (*JQODI* vs *Optimum*: 7651.3 J vs 6988.1 J, 142.335 s vs 135.564 s). This highlights the practical feasibility and efficiency of *JQODI*.

2) Figure 8 further shows the specific time and energy consumption of the feasible strategies when using network infrastructure. From Figure 8(a), we can see that for each strategy, the required computation time decreases as the amount of network infrastructure resources increases, i.e., $SD > ES > CS$. However, for energy consumption, the result is exactly the opposite, as shown in Figure 8(b), i.e., $SD < ES < CS$. These observations highlight the necessity of balancing energy consumption and computation time/time taken. In this sense, it is essential to carefully determine the optimal model partitioning point to achieve the most efficient distribution of computational tasks between SD and ES.

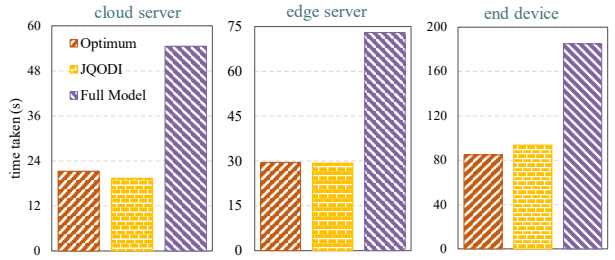
3) For inference performance, Figure 9 shows the inference overhead generated by the solutions corresponding to different strategies. We can see that our *JQODI* outperforms all other strategies in terms of both inference time and energy consumption across any individual device or device combination. This is because *JQODI* utilizes a lower bit-width than *Optimal* (i.e., 6-bit vs. 8-bit). We also observe that performing inference exclusively on Raspberry Pi5 offers a compelling option: When its inference time is higher than that of the combined PI+3090 setup, it still outperforms PI+4090. More notably, the energy consumption of Raspberry Pi5 inference is substantially lower than that of the combined inference approaches. However, *Full Model* is clearly undesirable, as the inference time of full-precision model solely on the Raspberry Pi5 reaches 18 seconds, significantly exceeding the acceptable time limit.

4) Figure 10 shows the time and energy consumed by different network infrastructures when adopting combinatorial inference. We find that for the inference of small models, the



(a) loss-energy curve (b) loss-time curve (c) time-energy curve

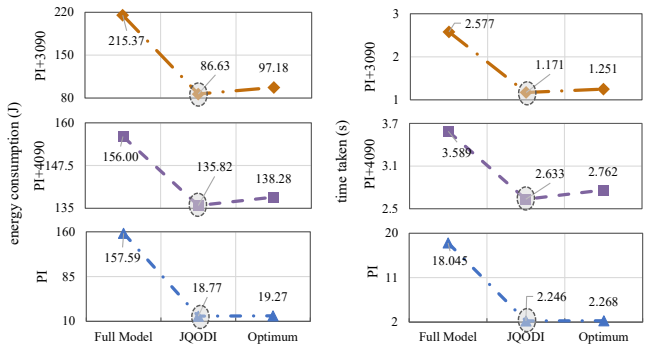
Fig. 7. The utility of benchmark strategies and JQODI based on Data 1: (a) the relationship between loss value and energy cost; (b) the relationship between loss value and time taken; (c) the relationship between time taken and energy consumption. The circle at the end of the line in (c) denotes the moment when the target loss $\mathcal{L}_{max}$ is reached, while the square at the end of the line indicates that this scheme cannot achieve the target loss $\mathcal{L}_{max}$ no matter how many epochs it is trained.



(a) time taken

(b) energy consumption

Fig. 8. The energy consumption and time taken on different network infrastructures during the training process of Data 1 using strategies *Optimum*, *JQODI*, *Full Model*, respectively.



(a) energy consumption (b) time taken

Fig. 9. The energy and time consumption required for inferring Data 1 on different devices, where PI+3090: infers shallow models on Raspberry Pi5 and deep models on RTX 3090; PI+4090: infers shallow models on Raspberry Pi5 and deep models on RTX 4090; PI: the entire model is reasoning on Raspberry Pi5.

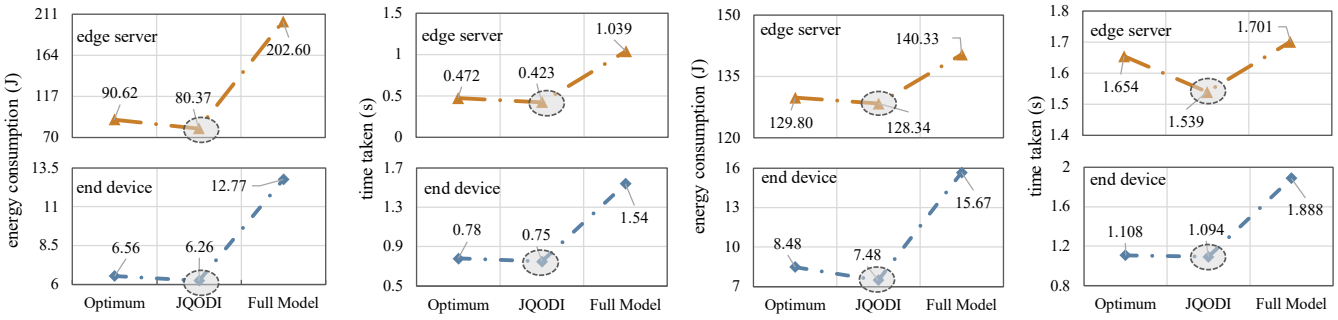
combination of 4090+PI is not an excellent choice. As shown in Figure 10(d), when using this combination for inference of small models, the time consumption of RTX 4090 even exceeds that of Raspberry Pi5. For the inference of large model, the combination of 3090+PI is suitable for delay sensitive situations, while the combination of 4090+PI is more suitable for our target situations that require lower energy consumption in this paper.

D. JQODI 's Performance on Data 2

1) We use Data 2 to simulate the situation of heterogeneous terminal data to explore the performance of *JQODI*. The *Minimal Model* is to select the second partitioning point in the VGG-like network with 1-bit quantization and deploy the deep model at RTX 4090; the *Optimum* is to select the first partitioning point in the ResNet20 network with 6-bit quantization and deploy the deep model at RTX 4090; the difference between *JQODI* and *Optimum* is that the quantization scheme adopted for the ResNet20 network is 8-bit; the results generated by *Full Model* correspond to the performance of the full-precision LeNet-like network when partitioning at the first point and deploying the deep model at RTX3090.

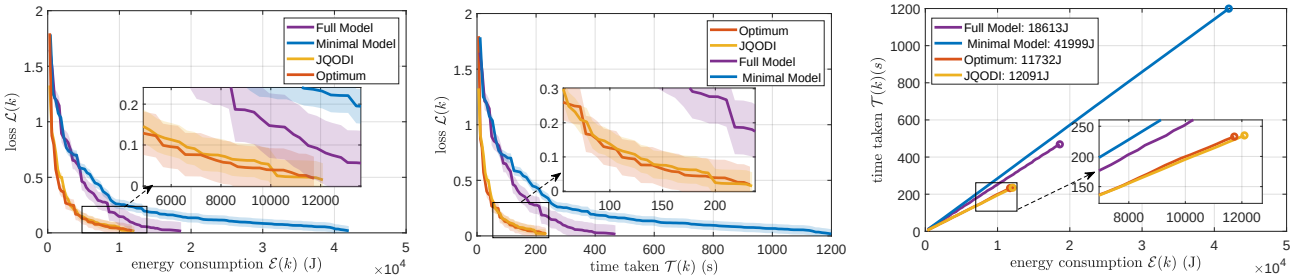
Figures 11(a)-(b) show the energy consumption and the time taken versus the loss value, respectively. We can observe that the result of *JQODI* is very close to the optimal result and outperforms the other strategies. Figure 11(c) shows the relationship between the time taken and the energy consumption of different strategies when achieving the target loss. As shown in Figure 11(c), for Data 2, all strategies are able to achieve the target loss of 0.02 within the prescribed time constraints. In contrast, for Data 1, the *Minimal Model* consistently fails to reach the target loss. Although *Full Model* does achieve the target loss, it requires significantly more time than is allowed by the specified constraints. We believe that the reason for this difference is that Data 2 has more training data compared to Data 1, which is more conducive for DNN to capture data features and achieve better training results.

2) Figure 12 illustrates the time and energy consumption of various strategies when utilizing the network infrastructure. It is evident that the *Minimal Model* strategy allocates the greatest proportion of time to SD, compared to the total



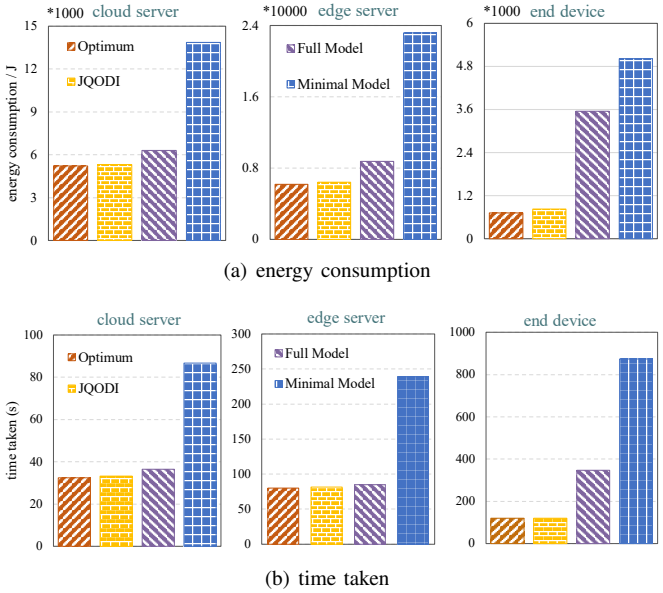
(a) PI+3090 (b) PI+3090 (c) PI+4090 (d) PI+4090

Fig. 10. The time and energy consumption on different network infrastructures during the inferring process of Data 1.



(a) loss-energy curve (b) loss-time curve (c) time-energy curve

Fig. 11. The utility of benchmark strategies and JQODI based on Data 2: (a) the relationship between loss value and energy cost; (b) the relationship between loss value and time taken; (c) the relationship between time taken and energy consumption.



(a) energy consumption

(b) time taken

Fig. 12. The energy consumption and time taken on different network infrastructures during the training process of Data 2 using strategies *Optimum*, *JQODI*, *Full Model*, and *Minimal Model*, respectively.

execution time. This outcome is due to the strategy's emphasis on executing tasks at the edge (partitioning point 2), which contrasts with the other strategies. When time constraints are more flexible, offloading a greater number of tasks to the SD can lead to a significant reduction in energy consumption.

VI. Related Work

This work involves the following research aspects, which are presented in detail below.

Model quantization. Its aim is to convert full-precision models to low-bit lightweight ones with no/minor performance degradation [24], [31], [36], [44], where, e.g., for 32-bit full precision, the 8-bit quantization refers to 75% compression ratio of model, and so forth. In this way, the model size as well as the power consumption can be reduced at the same time inference speed is accelerated, which is particularly important to the resource-constrained EC scenarios [18], [19], [36]. Early studies use post-training quantization (PTQ) [44], where the model weights or activations are trained and then quantized via low-bit approximations, but suffer from accuracy decrease. Then, quantization-aware training (QAT) is proposed [19], [31], aiming to train both the full-precision model and the quantization model simultaneously. In our previous paper [31], we propose an effective BQAT method using gradient backtracking compensation, and receives promising results, which is applied as quantization baseline in this work.

Model partitioning. It strives to enhance training efficiency and reduce memory consumption through dividing large models into a set of small segments being processed in parallel across multiple devices [14], [20], [43], [45]. Many partitioning methods have been proposed and developed to support collaborative execution under heterogeneous resource constraints [13], [25], [42], [45]–[47]. Especially, the model partitioning can be integrated with other strategies such as multi-exit mechanisms to address task latency constraints [46]. Similarly, the delay-aware DNN inference is realized via

TABLE VIII
A COMPARISON OF DIFFERENT HYBRID STRATEGIES MECHANISMS.

Mechanism	Properties			
	Compression	Partitioning	RA[1]	Joint-decision
[4]	✓	×	✓	✓
[23]	✓	×	×	×
[31]	✓	×	×	×
[46]	×	✓	✓	×
[48]	×	✓	✓	✓
[52]	×	×	✓	×
[55]	✓	×	✓	✓
JQODI	✓	✓	✓	✓

[1] Resource Allocation (RA)

jointly optimizing DNN partitioning and multi-thread parallelism (aiming to increase the number of admissible delay-aware DNN requests) [43]. In addition, SplitFed [47] integrated model partitioning with distributed learning paradigms, aiming to improve federated training efficiency and privacy through parallel collaborative model training. In this work, MCIA [13] is applied as baseline for model partitioning in cloud-edge-end collaboration.

Resource allocation. Pioneer attempts focus on auction-based resource allocation design [22], [49], [50], e.g., hierarchical auction [49], and smart contract-based double auction [50]. Recently, the game theory is introduced for task offloading and resource allocation [23], [52], [53], e.g., potential game [52], which validates the existence of a Nash Equilibrium in target system. In addition, the stochastic Stackelberg game [53] is often utilized to model the interactions between service providers (sellers) and miners (buyers) in blockchain-based systems. Lyu et al. [51] investigates the joint design of AI model pre-training and fine-tuning by optimizing communication and computational resources in edge networks. However, due to the resource limitation of heterogeneous SDs, it is still challenging to achieve efficient resource allocation in cloud-edge-end collaboration systems.

Hybrid strategies. Existing studies focus on solving one or another of the aforementioned decision problems [4], [48], [54], [55], but neglect the necessity for a unified optimization approach across these intricately intertwined decisions. For example, Malandrino et al. [4] seek to reduce energy consumption during DNN training by combining model pruning, data selection, and resource allocation. Lyu et al. [54] analyzes the impact of model compression on AI model performance in edge environments. Zeng et al. [48] leverage RL to identify the optimal partitioning method among available heterogeneous devices to accelerate the inference process. Zhang et al. [55] propose an cloud-edge collaboration framework based on RL and supervised learning to compress DNN models and utilize resource allocation to reduce task latency. To fill this gap, we introduce a joint-decision framework JQODI, which encompasses all the required decisions (formulated as a multi-dimensional optimization problem) through cloud-edge-end collaboration. Table VII summarizes the comparison between our JQODI and existing works.

VII. CONCLUSION

In this paper, we consider the optimization of collaborative inference and fine-tuning strategies for DNNs in EC scenarios via joint decision-making that concerns (i) model selection, (ii) model quantization, (iii) model partitioning, and (iv) resource allocation, which are tightly coupled with each other. To realize this, we propose a joint-decision solution JQODI to tackle the above decisions-coupled optimization for the minimization of energy consumption. This is the first attempt to address such problem. In particular, JQODI has polynomial worst-case complexity, and it provides near-optimal solutions by efficiently approximating the optimal solution for the NP-hard problem. Rigorous theoretical analysis and extensive numerical evaluations validate the effectiveness and efficiency of our JQODI in resource-limited EC scenarios, surpassing existing state-of-the-art methods and aligning closely with the optimal solution. Note that our current design primarily focuses on energy consumption. In the future, we plan to take into account latency and accuracy as additional optimization objectives, aiming to enhance the comprehensive performance of the entire system.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China under Grant 62472079.

REFERENCES

[1] Lyu Z, Gao Y, Chen J, et al. "Empowering Intelligent Low-altitude Economy with Large AI Model Deployment", 2025, [Online]. Available: https://arxiv.org/pdf/2505.22343
[2] Zhou W, Fan L, Zhou F, et al. "Priority-aware resource scheduling for UAV-mounted mobile edge computing networks", in *IEEE Transactions on Vehicular Technology*, vol. 72, no. 7, pp. 9682–9687, 2023.
[3] G. Quan, A. Eryilmaz and N. B. Shroff. "Minimizing Edge Caching Service Costs Through Regret-Optimal Online Learning", in *IEEE/ACM Transactions on Networking*, vol. 32, no. 5, pp. 4349-4364, 2024
[4] Malandrino F, Di Giacomo G, Karamzade A, et al. "Tuning DNN Model Compression to Resource and Data Availability in Cooperative Training", in *IEEE/ACM Transactions on Networking*, vol. 32, no. 2, pp. 1600–1615, 2023.
[5] S. Lin et al., "Leveraging Synergies Between AI and Networking to Build Next Generation Edge Networks", in *IEEE 8th International Conference on Collaboration and Internet Computing (CIC)*, pp. 16–25, 2022.
[6] Alahi MEE, Sukkuea A, Tina FW, et al. "Integration of IoT-Enabled Technologies and Artificial Intelligence (AI) for Smart City Scenario: Recent Advancements and Future Trends", in *Sensors*, vol. 23, no. 11, 2023.
[7] Ma L, Kang H, Yu G, et al. "Single-Domain Generalized Predictor for Neural Architecture Search System" in *IEEE Transactions on Computers*, vol. 73, no. 5, pp. 1400–1413, 2024.
[8] Lyu Z, Zhu G, Xu J, et al. "Semantic Communications for Image Recovery and Classification via Deep Joint Source and Channel Coding", in *IEEE Transactions on Wireless Communications*, vol. 23, no. 8, pp. 8388 – 8404, 2024.
[9] Hu M, Guo Z, Wen H, et al. "Collaborative Deployment and Routing of Industrial Microservices in Smart Factories", in *IEEE Transactions on Industrial Informatics*, vol. 20, no. 11, pp. 12758 – 12770, 2024.
[10] Bertino E, Banerjee S. "Artificial Intelligence at The Edge", 2020, [Online]. Available: https://arxiv.org/abs/2012.05410
[11] Wang H, Li Q, Kang H, et al. "ParaLoupe: Real-time Video Analytics on Edge Cluster via Mini Model Parallelization", in *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 13945–13962, 2024.
[12] Yang J, Guo Z, Luo J, et al. "Cloud-edge-end Collaborative Caching based on Craph Learning for Cyber-physical Virtual Reality", in *IEEE Systems Journal*, vol. 17, no. 4, pp. 5097–5108, 2023.

[13] Qi H, Ren F, Wang L, et al. "Multi-compression Scale DNN Inference Acceleration based on Cloud-edge-end Collaboration", in *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 1, pp. 1–25, 2024.

[14] Benelallam A, Tisi M, Cuadrado J S, et al. "Efficient Model Partitioning for Distributed Model Transformations", in *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*, pp. 226–238, 2016.

[15] Shlezinger N, Bajić I V. "Collaborative Inference for AI-empowered IoT Devices", in *IEEE Internet of Things Magazine*, vol. 5, no. 4, pp. 92–98, 2024.

[16] Parmezan A R S, Lee H D, Spolaôr N, et al. "Automatic Recommendation of Feature Selection Algorithms based on Dataset Characteristics", in *Expert Systems with Applications*, vol. 185, 2021.

[17] Malandrino F, Chiasserini C F, Di Giacomo G. "Efficient Distributed DNNs in the Mobile-Edge-cloud Continuum", in *IEEE/ACM Transactions on Networking*, vol. 31, no. 4, pp. 1702–1716, 2022.

[18] Gong R, Liu X, Jiang S, et al. "Differentiable Soft Quantization: Bridging Full-precision and Low-bit Neural Networks", in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4852–4861, 2019.

[19] Qu Z, Zhou Z, Cheng Y, et al. "Adaptive Loss-aware Quantization for Multi-bit Networks", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 7988–7997, 2020.

[20] Li H, Li X, Fan Q, et al. "Distributed DNN Inference with Fine-grained Model Partitioning in Mobile Edge Computing Networks", in *IEEE Transactions on Mobile Computing*, vol. 23, no. 10, pp. 9060–9074, 2024.

[21] Dong C, Hu S, Chen X, et al. "Joint Optimization with DNN Partitioning and Resource Allocation in Mobile Edge Computing", in *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 3973–3986, 2021.

[22] Ma L, Wang X, Wang X, et al. "TCDA: Truthful Combinatorial Double Auctions for Mobile Edge Computing in Industrial Internet of Things", in *IEEE Transactions on Mobile Computing*, vol. 21, no. 11, pp. 4125–4138, 2021.

[23] Wang X, Ye J, Lui J C S. "Decentralized Scheduling and Dynamic Pricing for Edge Computing: A Mean Field Game Approach", in *IEEE/ACM Transactions on Networking*, vol. 31, no. 3, pp. 965–978, 2022.

[24] Kim J, Chang S, Kwak N. "PQK: Model Compression via Pruning, Quantization, and Knowledge Distillation", 2021, [Online]. Available: https://arxiv.org/abs/2106.14681

[25] Jarachanthan J, Chen L, Xu F, et al. "Amps-inf: Automatic Model Partitioning for Serverless Inference with Cost Efficiency", in *Proceedings of the 50th International Conference on Parallel Processing*, pp. 1–12, 2021.

[26] X. Qin, B. Li and L. Ying. "Efficient Distributed Threshold-Based Offloading for Large-Scale Mobile Cloud Computing", in *IEEE/ACM Transactions on Networking*, vol. 31, no. 1, pp. 308–321, 2023.

[27] He K, Zhang X, Ren S, et al. "Deep Residual Learning for Image Recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[28] LeCun Y, Bottou L, Bengio Y, et al. "Gradient-based Learning Applied to Document Recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324 (Nov 1998)

[29] Krizhevsky A, Hinton G. "Learning Multiple Layers of Features From Tiny Images", https://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf

[30] Simonyan, K., Zisserman, A. "Very Deep Convolutional Networks for Large-scale Image Recognition", *3rd International Conference on Learning Representations*, 2015. https://doi.org/10.48550/arXiv.1409.1556

[31] Ma L, Zhou Y, Ma J, et al. "One-step Forward and Backtrack: Overcoming Zig-zagging in Loss-aware Quantization Training", in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 13, pp. 14246–14254, 2024.

[32] Shannon C E. "A Mathematical Theory of Communication", in *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[33] Saxe A M, McClelland J L, Ganguli S. "Exact Solutions to The Nonlinear Dynamics of Learning in Deep Linear Neural Networks", 2013, [Online]. Available: https://arxiv.org/abs/1312.6120

[34] Allen-Zhu Z, Li Y, Liang Y. "Learning and Generalization in Overparameterized Neural Networks, Going Beyond Two Layers", in *Advances in neural information processing systems*, vol. 32, 2019.

[35] Hestness J, Narang S, Ardalani N, et al. "Deep Learning Scaling is Predictable, Empirically", 2017, [Online]. Available: https://arxiv.org/abs/1712.00409

[36] Lee J, Kim D, Ham B. "Network Quantization with Element-wise Gradient Scaling", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6448–6457, 2021.

[37] Pisinger D. "Where Are The Hard Knapsack Problems", in *Computers & Operations Research*, vol. 32, no. 9, pp. 2271–2284, 2005.

[38] Pferschy U, Schauer J. "The Knapsack Problem with Conflict Graphs", in *Journal of Graph Algorithms and Applications*, vol. 13, no. 2, pp. 233–249, 2009.

[39] Morris J M. "Traversing Binary Trees Simply and Cheaply", in *Information Processing Letters*, vol. 9, no. 5, pp. 197–200, 1979.

[40] Osta M, Alameh M, Younes H, et al. "Energy Efficient Implementation of Machine Learning Algorithms on Hardware Platforms", in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 21–24, 2019.

[41] Zaw C W, Pandey S R, Kim K, et al. "Energy-aware Resource Management for Federated Learning in Multi-access Edge Computing Systems", in *IEEE Access*, vol. 9, pp. 34938–34950, 2021.

[42] F. Wang, S. Cai, V. K. N. Lau. "Decentralized DNN Task Partitioning and Offloading Control in MEC Systems With Energy Harvesting Devices", in *IEEE Journal of Selected Topics in Signal Processing*, vol. 17, no. 1, pp. 173–188, 2023.

[43] Li J, Liang W, Li Y, et al. "Throughput Maximization of Delay-aware DNN Inference in Edge Computing by Exploring DNN Model Partitioning and Inference Parallelism", in *IEEE Transactions on Mobile Computing*, vol. 22, no. 5, pp. 3017–3030, 2021.

[44] Liu Z, Wang Y, Han K, et al. "Post-training Quantization for Vision Transformer", in *Advances in Neural Information Processing Systems*, vol. 34, pp. 28092–28103, 2021.

[45] Dey S, Mondal J, Mukherjee A. "Offloaded Execution of Deep Learning Inference at Edge: Challenges and Insights", in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 855–861, 2019.

[46] Li E, Zeng L, Zhou Z, et al. "Edge AI: On-demand Accelerating Deep Neural Network Inference via Edge Computing", in *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[47] Thapa C, Arachchige P C M, Camtepe S, et al. "Splitfed: When Federated Learning Meets Split Learning", in *Proceedings of the AAAI conference on artificial intelligence*, vol. 36, no. 8, pp. 8485–8493, 2022.

[48] Zeng L, Chen X, Zhou Z, et al. "Coedge: Cooperative DNN Inference with Adaptive Workload Partitioning over Heterogeneous Edge Devices", in *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2020.

[49] A. Kiani and N. Ansari, "Toward Hierarchical Mobile Edge Computing: An Auction-based Profit Maximization Approach," in *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2082–2091, September 2017.

[50] Y. Du, Z. Wang, J. Li, and L. Shi, "Blockchain-Aided Edge Computing Market: Smart Contract and Consensus Mechanisms," in *IEEE Transactions on Mobile Computing*, vol. 22, no. 6, pp. 3193–3208, January 2022.

[51] Z. Lyu, Y. Li, G. Zhu, J. Xu, H. Vincent Poor and S. Cui, "Rethinking Resource Management in Edge Learning: A Joint Pre-Training and Fine-Tuning Design Paradigm," in *IEEE Transactions on Wireless Communications*, vol. 24, no. 2, pp. 1584–1601, 2025.

[52] Xu Z, Ren H, Liang W, et al. "Near Optimal Learning-driven Mechanisms for Stable NFV Markets in Multitier Cloud Networks", in *IEEE/ACM Transactions on Networking*, vol. 30, no. 6, pp. 2601–2615, 2022.

[53] A. Asheralieva and D. Niyato, "Learning-Based Mobile Edge Computing Resource Management to Support Public Blockchain Networks," in *IEEE Transactions on Mobile Computing*, vol. 20, no. 3, pp. 1092–1109, December 2019.

[54] Z. Lyu, M. Xiao, J. Xu, M. Skoglund, and M. Renzo, "The Larger the Merrier? Efficient Large AI Model Inference in Wireless Edge Networks," 2025, [Online]. Available: https://arxiv.org/pdf/2505.09214

[55] Zhang T, Li Z, Chen Y, et al. "Edge-cloud Cooperation for DNN Inference via Reinforcement Learning and Supervised Learning", in *2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)* pp. 77–84, 2022.

Yuee Zhou received the BSc degree from the Northeastern University, China, in 2017. She is currently pursuing the Ph.D. degree at Northeastern University, Shenyang, China. Her current research interests include edge computing and machine learning.

Lianbo Ma received the BSc and MSc degrees in communication and information system from Northeastern University, Shenyang, China, in 2004 and 2007, respectively, and the PhD degree from the University of Chinese Academy of Sciences, China, in 2015. He is currently a professor with Northeastern University. His current research interests include computational intelligence and machine learning. He has published more than 100 journal articles, books and refereed conference papers.

Xingwei Wang received the BSc, MSc, and PhD degrees from the Northeastern University, Shenyang, China, in 1989, 1992, and 1998, respectively, all in computer science. He is currently a professor at Northeastern University, China. His current research interests include cloud computing and future Internet. He has published more than 100 journal articles, books, and refereed conference papers.

Qing Li (Senior Member, IEEE) received the B.S. degree in computer science and technology from the Dalian University of Technology, Dalian, China, in 2008, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2013. He is currently an Associate Researcher with the Peng Cheng Laboratory, Shenzhen, China. His research focuses on network function virtualization, in-network caching/computing, intelligent self-running networks, and edge computing.

Carla Fabiana Chiasserini (Fellow, IEEE) is a Professor with the Electronics and Telecommunications Department of Politecnico di Torino, where she leads the Advanced Wireless Experience (AWE) Lab, and a Research Associate with the Italian National Research Council (CNR). She currently serves as the Vice Rector for Alumni and Career Orientation at Politecnico di Torino. She is an IEEE Fellow. Her research interests include the design of wireless network architectures, the definition of protocols and algorithms for mobile networks and applications, and the creation of theoretical models for the study and the performance evaluation of wireless systems. Her research is indeed a combination of system design and network-theory, aiming to design algorithms and methods that solve real problems, starting with actual data and scenarios. As far as application domains are concerned, she has been focusing on energy efficiency of wireless systems, connected cars, mobility services, service virtualization, and smart healthcare.

Guangjie Han (Fellow, IEEE) is currently a Professor with the Department of Internet of Things Engineering, Hohai University, Changzhou, China. He received his Ph.D. degree from Northeastern University, Shenyang, China, in 2004. In February 2008, he finished his work as a Postdoctoral Researcher with the Department of Computer Science, Chonnam National University, Gwangju, Korea. From October 2010 to October 2011, he was a Visiting Research Scholar with Osaka University, Suita, Japan. From January 2017 to February 2017, he was a Visiting Professor with City University of Hong Kong, China. From July 2017 to July 2020, he was a Distinguished Professor with Dalian University of Technology, China. His current research interests include Internet of Things, Industrial Internet, Machine Learning and Artificial Intelligence, Mobile Computing, Security and Privacy. Dr. Han has over 500 peer-reviewed journal and conference papers, in addition to 160 granted and pending patents. Currently, his H-index is 81 and i10-index is 381 in Google Citation (Google Scholar). The total citation count of his papers raises above 23300 times.