An Elastic Federated Learning Collaboration Framework for Computing-Constrained IoT

Haizhou Wang, Guobing Zou, Kun Cao, *Member, IEEE*, Yangguang Cui, *Member, IEEE*, Tongquan Wei, and Shiyan Hu

***Abstract*—Through exploiting decentralized data from multisource Internet of Things (IoT) devices, federated learning (FL) can accomplish the training of deep neural network (DNN) models in a privacy-preserving manner to provide premium intelligent services. Due to portability considerations, most IoT devices are computing-constrained which cannot afford frequent DNN model training in FL. Existing approaches use model compression techniques to reduce computing cost of IoT devices, whereas accuracy degradation is inevitably incurred. To address this challenge, we propose an elastic FL collaboration framework (EFLCF), namely, EFLCF, to accommodate limited computing resources of IoT devices. Specifically, we first design an FL-oriented elastic neural network model with multiple-width subnets, and couple it with an FL device-server collaboration framework to form EFLCF, thereby releasing computing cost pressure of IoT devices. We then develop a freezing-assisted wide-to-narrow training mechanism to realize efficient device-server distributed training and further reduce device computing cost. Finally, we design an entropy-based narrow-to-wide elastic inference mechanism to decrease computing cost of inference without compromising accuracy. Experiments demonstrate that compared to well-known benchmarks, our EFLCF can reduce up to 97.65% device computing cost and improve up to 48.3% accuracy in training, while reducing up to 42.5% computing cost in inference.**

***Index Terms*—Computing cost, federated learning (FL), inference mechanism, Internet of Things (IoT), training mechanism.**

Received 16 October 2024; revised 21 May 2025; accepted 5 July 2025. Date of publication 28 July 2025; date of current version 23 February 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62272290 and Grant 62272169; in part by the Natural Science Foundation of Shanghai under Grant 24ZR1421500; in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2024A1515010232; in part by the Open Research Fund of National Mobile Communications Research Laboratory (Southeast University) under Grant 2025D12; and in part by the Shanghai Trusted Industry Internet Software Collaborative Innovation Center. This article was recommended by Associate Editor W. Li. *(Corresponding author: Yangguang Cui.)*

Haizhou Wang, Guobing Zou, and Yangguang Cui are with the School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China (e-mail: ygcui@shu.edu.cn).

Kun Cao is with the College of Cyber Security, Jinan University, Guangzhou 510632, China, also with the National Mobile Communication Research Laboratory, Southeast University, Nanjing 211189, China, and also with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China.

Tongquan Wei is with the School of Computer Science and Technology, East China Normal University, Shanghai 200062, China.

Shiyan Hu is with the Department of Data and Systems Engineering, University of Hong Kong, Hong Kong, SAR.

Digital Object Identifier 10.1109/TCAD.2025.3593436

I. Introduction

With the prosperity of artificial intelligence, deep neural network (DNN) models have achieved tremendous success in computer vision [1], natural language processing [2], and other fields [3], enabling a large number of intelligent service applications. Meanwhile, as the information society advances, the number of Internet of Things (IoT) devices connected to the network is proliferating and capable of providing convenient services to humans [4]. Deploying these advanced DNN models on IoT devices can provide convenient intelligent services and yield tremendous socio-economic benefits [5]. However, there are two crucial issues for the efficient deployment of DNN models on IoT devices.

First, during the evolutionary development of artificial intelligence, most works focus on designing DNN models with more parameters and more complex structures in pursuit of superior performance [6]. However, while attaining superior performance, DNN models with numerous parameters and complex structures impose high computational overhead and runtime during their deployment [7], [8]. In addition, this problem is exacerbated by the fact that IoT devices are mostly computing-constrained. Oftentimes, to enable ubiquitous and convenient deployment in real-world scenarios, IoT devices are designed to meet portability requirements, which require them to be small, lightweight, and cost-effective [9]. These portability-driven hardware design features prevent IoT devices from accommodating powerful computing components, thereby making it difficult to support the computing-intensive training of DNN models. Thus, there is urgent need to efficiently deploy DNN models on computing-constrained IoT devices [10].

Second, with a wide range of IoT devices being deployed in various domains, vast amounts of data are collected by IoT devices during their deployment, and processing these multinode data usually need to train DNN models [11]. Traditionally, these collected data need to be transferred to one central server for DNN model training through extracting the information and knowledge within the data [12]. However, transferring raw data that contain sensitive user information, such as in the field of smart healthcare and finance, etc., induces data security and user privacy concerns. The recent issuance of relevant data protection regulations has rendered traditional centralized training difficult to implement.

To tackle these issues, federated learning (FL) emerges as a distributed DNN model training approach to exploit

TABLE I
EXAMPLE OF TRAINING LATENCY COMPARISON FOR COMPLETING A SINGLE TRAINING EPOCH

Device	Raspberry Pi 4B	Raspberry Pi 5	Jetson Nano B01
Latency	13987 s	5759 s	1107 s

decentralized data across multisource IoT devices in the absence of data privacy exposures [10], [13]. Specifically, one round of the training process for FL can be described as four main steps.

1) In the current round, participating clients, i.e., IoT devices, download a global DNN model from the parameter server.
2) Using their dataset, clients proceed in parallel to train this DNN model locally.
3) Clients pass their updated DNN models to the server.
4) Based on the *FedAvg* formula [14], the parameter server merges the DNN models originated from the clients as a fresh global DNN model for the next training round.

In FL, the server and clients iteratively process the above four steps until the target accuracy is attained, in which the data of clients are not exposed. Although FL can well handle the issues of user data and privacy leakage in centralized training of DNN models, clients need to undertake high computing cost locally for training DNN models in FL [15]. Following the FL workflow, with the same DNN model deployed on different devices, the training latencies of devices would exhibit remarkable differences due to the significant differences in device hardware capabilities [16]. Table I provides an example to illustrate this fact. Here, we adopt three common IoT devices, containing a Raspberry Pi 4B, a Raspberry Pi 5 and a Jetson Nano B01, to measure the training latency for completing a single training epoch. It can be observed that the computing-constrained Raspberry Pi 4B consumes 13987 s to complete a single training epoch, whereas the Jetson Nano B01 consumes only 1107 s to complete a single training epoch. Due to the synchronous nature of FL, the computing-constrained devices become a bottleneck that induces excessive distributed training latency and degrades the overall efficiency of the FL system. In the context of FL, such as microcontrollers and sensors, may not be able to afford the cost of training AI models or may induce high latency [17].

To address the above bottleneck, in this article, we propose an elastic FL collaboration framework (EFLCF) for computing-constrained IoT scenarios. The contributions of this article can be summarized as follows.

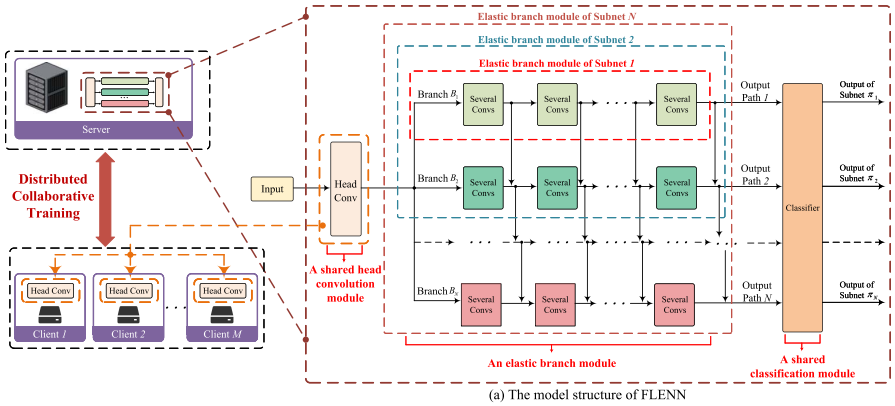
1) To release the pressure on computing cost of IoT devices, we propose an elastic FL collaboration framework, namely, EFLCF. In particular, this framework well couples our designed FL-oriented elastic neural network (FLENN) model with multiple-width subnets to realize the efficient device-server collaborative training.
2) To enhance training accuracy and further reduce computing cost of IoT devices, we develop a freezing-assisted wide-to-narrow training mechanism that can efficiently realize device-server collaborative training in EFLCF.
3) Through analyzing the FLENN model structure, we design an entropy-based narrow-to-wide inference mechanism that can realize elastic inference without sacrificing the model accuracy, thus decreasing the inference computing cost.
4) Extensive experiments demonstrate that using the well-known benchmarks, our EFLCF can improve training accuracy by up to 48.3%, reduce device computing cost by up to 97.65% in training, while reducing computing cost by up to 42.5% for achieving similar accuracy in inference.

II. RELATED WORK

In the last few years, several researchers have concentrated on optimizing the computing cost of DNN models when deployed in IoT devices. From the perspective of DNN model compression design, Howard et al. [18] proposed a depthwise separable convolution, a lightweight technique designed to replace the traditional computing-intensive convolution, to build an efficient DNN model, called MobileNet, which can effectively reduce computing cost. Zhao et al. [19] designed an uncertainty-aware binary neural network that utilizes a single bit rather than a floating point number to store model parameters, resulting in remarkable computing cost optimization. In spite of the fact that these model compression design schemes release some computing cost, the accuracy of DNN models appears to be degraded. Considering the aspect of the training mechanism design, Cai et al. [20] introduced a progressive shrinking training technique to train a super DNN model that contains myriad subnets with different scales and accuracy. Despite the effectiveness, huge computing cost is incurred during the training process of the super DNN model on IoT devices.

In addition, some researchers adopt the computing offloading idea to optimize computing cost of DNN models on IoT device deployments. Through collaboration between IoT devices and a server, Eshratifar et al. [21] designed the JointDNN algorithm which offloads computation to the server at the layer granularity of DNN models for optimizing latency. Further considering load balancing, Xu et al. [22] presented a distributed auction algorithm with polynomial complexity to obtain the optimal device-server computing offloading solution. Nevertheless, data characteristics are not taken into account in the above schemes, resulting in some easy-to-handle samples still being processed through the entire DNN model. To solve the above problem, Teerapittayanon et al. [23] developed a BranchyNet that contains a side-branch classifier in its early layers, thus enabling some easy-to-handle samples to exit from the side branch with less computing cost. Considering the reliability guarantee, Pang et al. [24] established a runtime adaptation framework that schedules DNN models with different computing cost to meet the real-time requirements. However, the above approaches only focus on optimizing computing cost in inference, while ignoring user privacy and data security in training.

To handle the above problem, FL enables distributed training of DNN models in a privacy-preserving fashion. Recently, considerable research efforts have been devoted to optimizing the computing cost of participating clients in FL. From the



(a) The model structure of FLENN

Fig. 1. Key components of our proposed EFLCF.

perspective of DNN model design, Shlezinger et al. [25] employed quantization theory tools to develop a Universal Vector Quantization technique, which realizes quantization compression of DNN models. Li et al. [26] designed a flexible compression scheme for DNN models which incorporates the Benders decomposition technique to reduce the resource consumption. Nevertheless, accuracy degradation is incurred in model compression schemes for FL. Cui et al. [17] designed a lightweight DNN model with one side branch to accommodate the resource-constrained nature of IoT devices in FL. In FL, Chen et al. [27] categorized the employed DNN models into the shallow part and deep part, and completed updates of two parts with different frequencies in FL to reduce training cost. However, resource management design of FL is not taken into account.

From the perspective of resource management strategy design, Zhan et al. [28] designed a deep reinforcement learning-based CPU-operation frequency control technique to reduce clients' computing cost in FL. In a time-sharing communication fashion, Tran et al. [29] formulated the trade-off problem of FL training delay and resource consumption as an optimization problem, and gave a closed-form solution for resource allocation through quantitative analysis. Wang et al. [30] modeled the resource allocation problem of FL as a Markov decision process and proposed a corresponding multiagent deep reinforcement learning strategy to dynamically accomplish resource allocation. Further consideration of client selection strategy design, [15] introduced a utility-enabled heterogeneous aware client selection strategy for FL to match communication resources. Chen et al. [12] designed a probabilistic client selection scheme coupled with an integer linear programming-based communication resource allocation scheme to accelerate the FL distributed training rate. However, the above works do not involve the DNN model design in FL frameworks, and thus remain unadaptable to IoT devices with extremely limited computing resources.

III. Overview of Our Proposed Approach

A. Components of Our EFLCF

Nowadays, the dynamic neural network design [8], [31] has become popular due to its elastic inference capability, which enables it to flexibly adapt to the resource-constrained nature and accuracy requirements of IoT intelligent applications. Inspired by the above facts, this work aims at exploring the potential benefits of the dynamic neural network paradigm in FL scenarios, and propose an FLENN model by further design optimization and secondary exploitation on the state-of-the-art multibranch scalable neural network (MBSNN) in [8].

To be specific, our proposed FLENN model structure is composed of three modules, i.e., a shared head convolution module, an elastic branch module, and a shared classification module, as shown in Fig. 1(a). First, different from the original MBSNN model, our proposed FLENN model structure adopts a stable and shared head convolution scheme to accommodate multiclient distributed training collaboration of FL. Second, in the elastic branch module, our FLENN model structure contains N branches $\{B_1, B_2, \ldots, B_N\}$ which can be elastically formed into its N subnets with different widths. More specifically, the elastic branch module of subnet π_1 refers to branch B_1, and the elastic branch module of subnet π_2 consists of branch B_1 and branch B_2. In general terms, the elastic branch module of subnet π_n is composed of the front n branches $\{B_1, B_2, \ldots, B_n\}$ and owns the corresponding width. Here, the subnet number of the FLENN model and the width of each subnet are determined by a width factor list $\{w_1, \ldots, w_n, \ldots, w_N\}(0 < w_1 < \ldots w_n < \ldots < w_N)$. A width factor w_n can adjust the filter number of convolutional layers on the branch module in subnet π_n. For instance, if $w_n = 0.5$, then the filter number in subnet π_n is reduced to half of the default number. Third, the output tensor of the elastic branch module with different width factors is passed to the shared classification module, and activates the corresponding part of neurons on the shared classification module for final classification results.

Leveraging the above FLENN model, we further design an EFLCF, namely, EFLCF, toward computing-constrained IoT devices, as illustrated in Fig. 1. Specifically, in this framework, the head convolution module, the first module of the FLENN model, is placed on the clients to reduce their computing cost in FL training, whereas the entire FLENN model is deployed on the server to ensure complete training for the FLENN model. In this framework, the clients only need to undertake the computing cost of the head convolution module of the FLENN model, and upload the output tensor of the head convolution

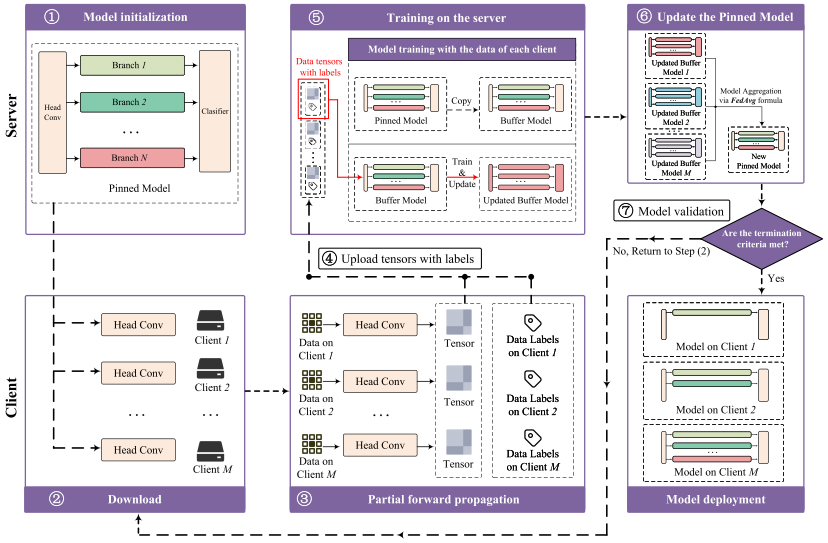


Fig. 2. Workflow of our proposed EFLCF.

module and data labels to the server for subsequent training. In this way, except for the head convolution module, the clients do not need to perform the forward and backward propagation required for training the FLENN model, which greatly reduces the computing cost of clients. Meanwhile, in our designed EFLCF, the clients only upload the feature tensor computed by the head convolution module without uploading the raw data, thus ensuring privacy protection.

B. Workflow of the EFLCF

This section describes the workflow of our proposed EFLCF, as shown in Fig. 2. In this framework, the server maintains two categories of models with the same FLENN model structure, which are the Pinned Model and the Buffer Model. Here, the Pinned Model records the initial parameters of the global model at the beginning of each training round, and the Buffer Model stores the model parameters after each training update via using a single client's data. Specifically, the workflow of each training round in our framework can be summarized in eight steps.

Step (1): The server completes the initialization of the Pinned Model.

Step (2): Participating clients download the head convolution of the Pinned Model from the server in the current training round.

Step (3): Participating clients utilize their local data to perform partial forward propagation by using the received the head convolution, thereby obtaining the low-level output tensor of training data.

Step (4): Participating clients transmit the obtained low-level output tensor of the training data and the corresponding data labels to the server.

Step (5): The server first replicates the current Pinned Model to obtain a Buffer Model, and then adopts the low-level output vectors and data labels transmitted by each client to train and update the Buffer Model parameters.

Step (6): Based on the *FedAvg* formula [13], the server integrates all updated Buffer Models obtained from participating clients to obtain a new Pinned Model for next training round.

Step (7): Check whether the current Pinned Model converges or meets the desired accuracy requirements. If so, collaborative training terminates; otherwise, return to Step (2) for the next training iteration.

After finishing the client-server collaborative training, each client elastically chooses to download and deploy either a partial Pinned Model or the entire Pinned Model depending on the accuracy requirements of its intelligent application.

C. Preliminaries

In this section, we illustrate the differences between the proposed FLENN model and the existing MBSNN model in terms of the deployment method, model structure, and training mechanism. Meanwhile, we explain why these differences can enable the proposed FLENN model to better improve the FL training efficiency for computing-constrained devices.

First, as far as the deployment method is concerned, the entire MBSNN model is deployed on a single IoT device. In contrast, the proposed FLENN model deploys its head convolution module on IoT devices and its remaining modules on a server, as shown in Fig. 1. This collaborative design in the deployment method allows computing-constrained devices to efficiently offload the computing burden of high-intensity training tasks to the powerful server. Meanwhile, through the collaboration of FL, this deployment method design can aggregate more diverse data from computing-constrained devices, thus improving the accuracy of the global model.

Second, with respect to the model structure design, the proposed FLENN model employs a shared head convolution module, while the head module of the MBSNN model employs a multiwidth structure that matches N branches of its elastic branch module. This improvement in the model structure design can reduce the computing overhead of devices in

Algorithm 1: Our Proposed EFLCF for Computing-Constrained IoT

Input: A server, a participating client set Θ of size K, a width factor list $\{w_1, w_2, \ldots, w_N\}$, a client selection fraction α, a maximum number of FL training rounds E;
Output: The well-trained global FLENN model;
1 Based on the given width factor list $\{w_1, w_2, \ldots, w_N\}$, the server initializes the Pinned Model G_P^1 with the FLENN model structure;
2 **for** $e \leftarrow 1, 2, \ldots, E-1$ **do**
3 Randomly select $\alpha \times K$ clients from the all client set Θ to construct the selected client set Φ for the current eth round;
4 The server sends the head convolution module of its current Pinned Model G_P^e to the participating client set Φ;
// Clients execute
5 **for** *Each participating client* $c_m \in \Phi$ ***in parallel*** **do**
6 By using the received head convolution module with its local data, client c_m performs partial forward propagation and obtains the corresponding low-level tensor v_m;
7 Client c_m uploads the obtained low-level tensor v_m of the training data and data labels l_m to the server;
8 **end**
// Server executes
9 **for** *Each participating client* $c_m \in \Phi$ **do**
10 For each client c_m, the server replicates a Buffer Model $G_{B,m}^e$ from its current Pinned Model G_P^e;
11 The Algorithm 2 is called on the server to utilize the tensor v_m and data labels l_m uploaded by the client c_m for training the Buffer Model $G_{B,m}^e$ as $G_{B,m}^{e+1}$;
12 **end**
13 The server integrates M Buffer Models $\{G_{B,1}^{e+1}, \ldots, G_{B,M}^{e+1}\}$ to obtain a new Pinned Model G_P^{e+1} as $G_P^{e+1} = \frac{\sum_{m=1}^{|\Phi|} |D_m| G_{B,m}^{e+1}}{\sum_{m=1}^{|\Phi|} |D_m|}$;
14 **if** *Pinned Model* G_P^{e+1} *is converged* **then**
15 **Return** The trained Pinned Model G_P^{e+1};
16 **end**
17 **end**
18 **Return** The trained Pinned Model G_P^{E+1};

Algorithm 2: Our Freezing-Assisted Wide-to-Narrow Training Mechanism for EFLCF

Input: Data tensor v_m with data labels l_m from the client c_m, a Buffer Model $G_{B,m}^e$ with the FLENN structure, a freezing milestone index F_s, a freezing gap value F_g, the current training epoch index e;
Output: The trained Buffer Model $G_{B,m}^{e+1}$ in this training round;
1 Server constructs optimizers $\{opt_1, \ldots, opt_N\}$ for updating parameters of the elastic branch module and shared classification module of N subnets $\{\pi_1, \ldots, \pi_N\}$ in the current Buffer Model $G_{B,m}^e$;
2 **for** $n \leftarrow N, N-1, \ldots, 1$ **do**
3 **if** $n < N$ *and* $n > 1$ *and* $rand() \leq 0.5$ **then**
4 continue;
5 **end**
6 The nth subnet clears its gradient information in Buffer Model, $opt[n].zero\_grad()$;
7 The nth subnet perform forward propagation, $\mathbf{y}_{\pi_n}^{\mathbf{m}} = \pi_n(\mathbf{v}^{\mathbf{m}})$;
8 Server obtains the training loss of the nth subnet by using (1);
9 Server updates model parameters of subnet π_n except for the head convolution module, $opt[n].step()$;
10 **if** $n == N$ *and* $(e \leq F_s$ *or* $e \,\%\, F_g == 0)$ **then**
11 The server conveys the client c_m the input vector for the head convolution in the back propagation;
12 Client c_m updates the head convolution module and sends it to the server;
13 Server adopts the received new head convolution module to replace the old one;
14 **end**
15 **end**
16 **Return** The updated Buffer Model $G_{B,m}^{e+1}$ in this training round;

FL. Specifically, in MBSNN with N subnets, each training sample is processed N times through the corresponding parts of its head module to generate N low-level feature tensors for training the subnets. In contrast, in FLENN with N subnets, each training sample only needs to be processed once through the shared head convolutional module to generate a unified low-level feature tensor that can be reused by all subnets.

Third, in terms of the training mechanism, the proposed FLENN model features a novel freezing-assisted wide-to-narrow training mechanism, and this mechanism is significantly different from the traditional training method of the MBSNN model. Here, this training mechanism utilizes a freezing technique to alternate between nonfreezing and freezing training rounds, and thus reduces the updating frequency of the head convolution on clients without compromising accuracy. As a consequence, based on this mechanism, our proposed EFLCF can further reduce the training overhead of computing-constrained devices. Note that the design details of our proposed training mechanism are provided in Section V.

IV. Our Proposed EFLCF for Computing-Constrained IoT

Algorithm 1 elaborates on the implementation details of our proposed EFLCF for computing-constrained IoT devices. First of all, by adopting a set width factor list, this algorithm initializes the Pinned Model with the FLENN model structure as a global model in line 1. Our framework can then perform iterative FL training via cooperation between a server and participating clients (lines 2–17).

Specifically, in each server-client collaborative training iteration, based on a given client selection factor, the server first randomly decides the participating clients for this training round in line 3, and assigns the head convolution of the current Pinned Model to these clients in line 4. Afterwards, the participating clients first process the local data in parallel using the received head convolution module to obtain the low-level output tensor of training data, and upload the obtained output tensor and the corresponding data labels to the server for subsequent training in lines 5–8. Once the low-level tensor and labels of the training data are received, the server starts performing subsequent training for each client (lines 9–12). That is, for each client, the server duplicates the current Pinned Model as a Buffer Model in line 10, and then calls Algorithm 2 to complete subsequent training of the Buffer Model using the low-level tensor and labels of the training data in line 11. After the server has processed the training of all participating clients, the server integrates all trained Buffer Models via the *FedAvg* formula [13] to obtain a new Pinned Model for the next distributed training iteration in line 13. Finally, the server checks whether the newly integrated Pinned Model has converged. If so, the algorithm returns the new Pinned Model from this training and terminates; otherwise, it proceeds to the next distributed training iteration (lines 14–16).

As illustrated in Algorithm 1, there is a clear difference between our proposed EFLCF and the classical FL framework in terms of the training pattern. In most of the classical FL

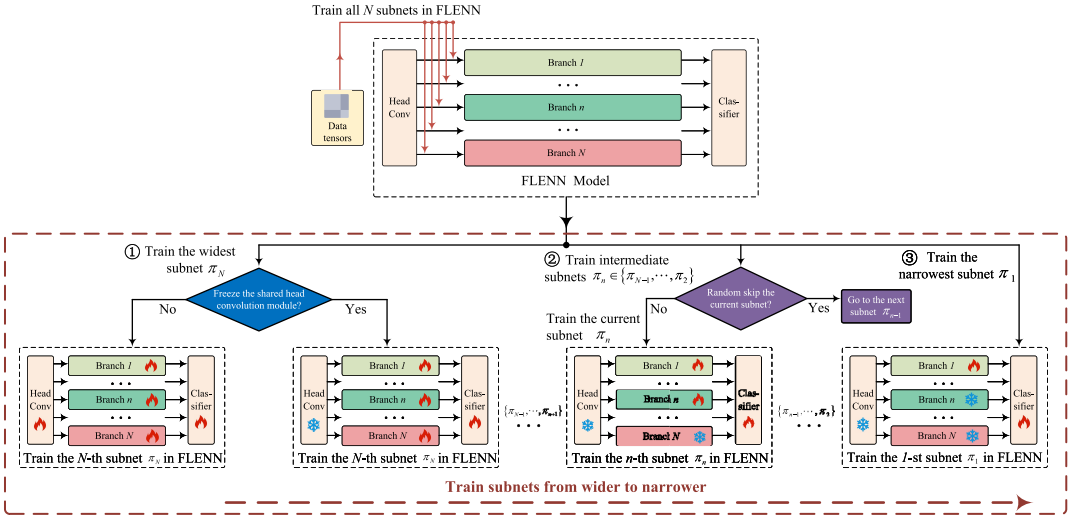


Fig. 3. Our designed freezing-assisted wide-to-narrow training mechanism for EFLCF.

frameworks, the clients are required to perform the complete computing-intensive DNN model training locally. In contrast, in our framework, clients need to undertake only a small computing cost for executing the head convolution module of the FLENN model, thereby greatly releasing their computing pressure. Meanwhile, our framework can yield a trained FLENN model which enables resilient inference. This feature of elastic inference means that the trained FLENN can be fully or partially deployed on IoT devices with different computing resources to perform the required inference requirements without fine-tuning, which can well broaden the application scenarios of FL and improve the usability of computing-constrained IoT devices for FL-driven intelligent applications.

V. Our Designed Freezing-Assisted Wide-to-Narrow Training Mechanism for EFLCF

A. Design Principles

Currently, most of the traditional training mechanisms are designed for classical DNN models with only one exit [1], [32], [33]. Therefore, these traditional mechanisms are not suitable for elastic neural networks with multiple exits [8], [31], [34]. To address the above challenges, several training techniques for elastic neural networks have been designed in the last few years. For instance, a common training technique [23] is to superimpose the losses of all the output exits in the elastic neural network, and updates all model parameters of the elastic neural network in a single back propagation. However, the existing training techniques for elastic neural networks are mainly designed for single-node IoT devices, and thus cannot be directly applied to FL distributed training paradigms.

Inspired by the above facts, we design a freezing-assisted wide-to-narrow training mechanism to well couple the FLENN model with multiple-width subnets into our proposed EFLCF, as shown in Fig. 3. Unlike existing training mechanisms [23], [35], our designed training mechanism calculates the loss and completes the model parameter update for each subnet $\pi_n (1 \le n \le N)$ in the FLENN model separately, that is

$$L\left(\mathbf{y}_{\pi_\mathbf{n}}; \hat{\mathbf{y}}; \theta_n\right) = -\hat{\mathbf{y}}^T \log\left(\mathbf{y}_{\pi_\mathbf{n}}\right)$$
$$\mathbf{y}_{\pi_\mathbf{n}} = \theta_n(\mathbf{x}), 1 \le n \le N \tag{1}$$

where $\mathbf{x}$ is input data tensors, $\hat{\mathbf{y}}$ is ground truth data labels, $\mathbf{y}_{\pi_\mathbf{n}}$ refers to output predicted results of the nth subnet π_n, θ_n represents parameters of the nth subnet π_n, and N is the number of subnets in the FLENN model. In this manner, it is observed that the parameters of the wide subnets encompass those of the narrow subnets. Therefore, when the wide subnets update their parameters, the parameters of the narrow subnets are also updated simultaneously. In this context, this work adopts the wide-to-narrow fashion to sequentially complete training for all subnets of the FLENN model in EFLCF. It is worth highlighting that in the wide-to-narrow fashion, updating the parameters of the wide subnets produces a positive pretraining function on the training of the narrow subnets, thereby better improving the training accuracy.

Furthermore, in our proposed EFLCF, all subnets share a head convolution module, and this shared head convolution module is placed on the client side for extracting low-level features from raw data. Obviously, updating the head convolution module imposes additional computing overhead on participating clients. To further release the computing pressure on clients, we develop a freezing strategy to lower the update frequency of the shared head convolution module, as elaborated in Algorithm 2. Note that since the shared head convolution module has only a few parameters and is used to extract the low-level tensor from raw data, it can still be trained well with less frequent updates controlled by our designed freezing strategy.

B. Freezing-Assisted Wide-to-Narrow Training Algorithm

Algorithm 2 shows the implementation details of our freezing-assisted wide-to-narrow training mechanism for EFLCF. This algorithm consists of a training setting initialization phase (line 1) and a multiple-subnet training phase (lines 2–15). In the initialization phase, each subnet π_n in the

Buffer Model is equipped with a corresponding optimizer opt_n to subsequently update its parameters of the elastic branch module and shared classification module in line 1.

In the multiple-subnet training phase, the subnets of the Buffer Model are updated sequentially in a wide-to-narrow fashion (lines 2–15). Here, when the wide subnets update its parameters, the parameters of the narrow subnets are also updated simultaneously to achieve the pretraining function. In the wide-to-narrow fashion, the training process for a subnet contains the random skip (lines 3–5), partial forward propagation (lines 6–8), partial backward propagation (line 9), and optional remaining backward propagation (lines 10–14). In the random skip, the intermediate subnet $\pi_n (1 < n < N)$ skips training with a 50% probability in lines 3–5, providing the regularization function for training. Note that the widest subnet π_N and smallest subnet π_1 are not skipped since they ensure the upper and lower bound performance of the FLENN model, respectively. In the partial forward propagation, the subnet π_n clears its optimizer gradient information in line 6, utilizes the data tensor v_m from the client c_m to complete the partial forward propagation and obtain the prediction result $\mathbf{y}^{\mathbf{m}}_{\pi_\mathbf{n}}$ in line 7, and computes the loss between the prediction result $\mathbf{y}^{\mathbf{m}}_{\pi_\mathbf{n}}$ and the ground truth label $\mathbf{y}^{\mathbf{m}}_{\pi_\mathbf{n}}$ by using (1) in line 8. In the partial backward propagation, the subnet π_n utilizes its optimizer opt_n to update the parameters of the corresponding elastic branch module and shared classification module in line 9. Finally, an optional remaining backward propagation for the head convolution module is processed in lines 10–14. In this phase, it is set that the head convolution module is updated only during the training of the widest subnet π_N. This is because the head convolution is a starting and few-parameter module for processing the original input data, and can be trained well even if it is frozen during the training of other subnets. Moreover, to further simplify the training process, we set a principle for reducing the update frequency of the head convolution module. That is, once the current training epoch index e exceeds the preset freeze milestone F_s, the update frequency of the head convolution is adjusted to once every F_g training rounds.

VI. Entropy-Based Narrow-to-Wide Elastic Inference Mechanism

A. Design Principles

Following the content of Sections IV and V, we can obtain a well-trained FLENN model via client-server collaborative training in EFLCF. In contrast to the traditional DNN models with one exit, the well-trained FLENN model contains multiple executable subnets, each of which has different model parameters and inference accuracy.

To match the varying difficulty of inferring data and the diverse accuracy requirements of FL intelligent applications, our designed FLENN model employs an entropy-based confidence threshold judgement mechanism to incrementally select subnets for elastic inference. For the detailed rationale of the entropy confidence judgement, the readers can refer to the literature [36]. Specifically, the entropy confidence level of subnets in the FLENN model is calculated as follows:

Algorithm 3: Our Entropy-Based Narrow-to-Wide Elastic Inference Mechanism

Input: A client c_m, a server, a well-trained FLENN model with N subnets $\{\pi_1, \pi_2, \ldots, \pi_N\}$, the input data x, a preset confidence threshold set $T = \{\tau_1, \tau_2, \ldots, \tau_N\}$;
Output: The inference result $\mathbf{y}$;
1 Client c_m extracts the low-level tensor x' by using the shared head convolution of FLENN model, $x' = head\_conv(x)$;
2 **for** $n \leftarrow 1, 2, \cdots N$ **do**
3 **if** *client c_m is able to perform inference of subnet π_n* **then**
4 FLENN model on client c_m switches to nth subnet π_n;
5 The nth subnet π_n on client c_m performs inference, $\mathbf{y} = \pi_n(x')$;
6 Client c_m obtains entropy confidence value v of inference result $\mathbf{y}$ by using (2);
7 **if** $v \leq \tau_n$ *or* $n == N$ **then**
8 **Return** Local inference result $\mathbf{y}$ on client c_m;
9 **end**
10 **else**
11 Call **Procedure** Sever_Inference (x', n, T) to conduct collaborative inference on the server;
12 **end**
13 **end**

14 **Procedure** Sever_Inference (x', n, T)
15 **for** $j \leftarrow n, \ldots, N$ **do**
16 FLENN model on the server switches to jth subnet π_j;
17 The jth subnet π_n on the server performs inference via the low-level tensor x' from client c_m, $\mathbf{y} = \pi_j(x')$;
18 Server calculate entropy confidence value v of inference result $\mathbf{y}$ by using (2);
19 **if** $v \leq \tau_j$ *or* $n == N$ **then**
20 **Return** Inference result $\mathbf{y}$ obtained by the server;
21 **end**
22 **end**

$$\tau(\mathbf{y}) = \sum_{i=1}^{l} \frac{y_i \log_2 y_i}{\log_2 l} \tag{2}$$

where $\mathbf{y}$ is the output vector of a subnet in the FLENN model, l denotes the length of the output vector $\mathbf{y}$, and y_i is the ith element of the vector $\mathbf{y}$ and represents the classification possibility of the ith class. Here, if the confidence level of a subnet is less than a predefined threshold, the obtained inference result of the current subnet is considered to be confident and the inference is terminated; otherwise, the next wider subnet in the FLENN model is employed for further inference. Based on the defined confidence level, we design a narrow-to-wide elastic inference mechanism, which will be elaborated in Algorithm 3. Benefiting from the above elastic inference mechanism, the FLENN model can utilize narrow subnets with low computing cost to handle simple samples and satisfy low-accuracy requirements, while relying on wide subnets with high computing cost to handle difficult samples and satisfy high-accuracy requirements. Moreover, consider that IoT devices may be computing constrained and not be able to afford the inference cost of some wide subnets in the FLENN model. Thus, to address this challenge, our inference mechanism also offers an optional server-dependent inference scheme without sacrificing privacy and exposing raw data.

B. Narrow-to-Wide Elastic Inference Algorithm

Algorithm 3 details the workflow of our entropy-based narrow-to-wide elastic inference algorithm. First of all, the

client c_m feeds the input data x into the shared-head convolution module of the trained FLENN model to obtain the low-level tensor x' in line 1. The algorithm then performs entropy-based elastic inference in a narrow-to-wide fashion (lines 2–13). At the beginning of the inference for the current subnet π_n, the client c_m evaluates whether it can afford the computing cost of performing the inference in line 3. If so, the client performs the inference locally in lines 4–10; otherwise, the client employs the server-dependent inference scheme via computing offloading in line 11. More specifically, in the client-side elastic inference process, the FLENN model on client c_m first switches to the current subnet π_n, and executes inference for the tensor x' to obtain an inference result $\mathbf{y}$ from subnet π_n in lines 4–5. The client c_m then calculates its inference result entropy confidence value v for the obtained inference result $\mathbf{y}$. Finally, the confidence value v is compared with the current subnet's confidence threshold value τ_n in lines 7-9. If so, the inference result $\mathbf{y}$ is confidently returned and the inference terminates in line 8; otherwise, the algorithm proceeds to the next subnet until all subnets are visited. In addition, when the client c_m cannot afford the inference cost for the current subnet π_n, the algorithm calls the Procedure Server_Inference to accomplish subsequent elastic inference on the server. Similar to the client-side elastic inference process, the Procedure Server_Inference is specified in lines 14–22. Note that the tensor x' uploaded by the client c_m is already processed locally in its shared convolution module, which means the raw data x can be firmly kept in the client with privacy-preserving features. Through using the above entropy-based narrow-to-wide elastic inference algorithm, computing-constrained clients can accomplish efficient and low-cost elastic inference in FL-enabled applications.

VII. Evaluation

A. Experimental Settings

In this work, we consider a common collaborative FL framework which consists of a server and 100 participating clients [37]. Our proposed EFLCF and mechanisms are all implemented by using Python 3.6.9 and PyTorch 1.4.0 and simulation experiments are conducted on a server with a Nvidia GTX 4090 GPU [36]. With respect to FL-related parameters, 10% of participating clients are randomly selected in each FL training round, and training samples are randomly shuffled and evenly distributed to 100 clients [11]. We adopt widely used Adam Optimizer [38] for FL collaborative training. Here, the maximum number of FL training rounds is set to 200, the learning rate is deemed to be 0.1, the weight decay rate is set to 0.0005, and the momentum is given as 0.9.

To verify the effectiveness of our proposed FLENN model, we adopt the well-known ResNet-18 [1] and MobileNetV2 [32] as backbone DNN models, respectively. Note that our FLENN model structure can be applied to many common DNN models, such as ResNet18 and MobileNetV2. Here, we adopt the style of adding a suffix after the DNN model name, i.e., *X-FLE-[A,B,C]*, to indicate that this *X* DNN model is coupled with the FLENN model structure, in which *[A,B,C]* represents that this model contains three subnets of widths [A,B,C]. For instance, *ResNet18-FLE-[0.3,0.6,1.0]* indicates one ResNet18 model with our FLENN model structure that contains three subnets of widths [0.3,0.6,1.0].

B. Performance Evaluation of Our FLENN Structure in Terms of Accuracy, Computing Cost, and Parameters

In this section, the performance of the FLENN model structure in our proposed EFLCF is evaluated. Here, we adopt ResNet18 and MobileNetV2 as backbone models to construct multiple corresponding FLENN models, including MobileNetV2-FLE-[0.1,0.2,0.3], MobileNetV2-FLE-[0.3,0.6,1.0], ResNet18-FLE-[0.1,0.2,0.3], and ResNet18-FLE-[0.3,0.6,1.0]. These models with the FLENN structure are compared with their primitive DNN models with maximum width, respectively. Note that the symbol MobileNetV2-0.3 refers to MobileNetV2 with a single 0.3 width path, and the symbols MobileNetV2-1.0, ResNet18-0.3, and ResNet18-1.0 characterize similar meanings. Performance is evaluated on CIFAR-10 and Fashion-MNIST datasets in terms of three metrics: accuracy, computing cost, and number of model parameters, as shown in Fig. 4.

Fig. 4(a)–(d) validate the accuracy improvement of MobileNetV2- and ResNet18-derived FLENN models on the CIFAR-10 and Fashion-MNIST datasets. Clearly, for the MobileNetV2- and ResNet18-derived FLENN models with multiple subnets of different widths, the accuracy of the subnet increases steadily with width. This is because wider subnets with more parameters have stronger feature extraction capabilities that can help them achieve higher accuracy. In particular, these MobileNetV2- and ResNet18-derived FLENN models all achieve higher accuracy compared to the their primitive DNN models with maximum width, respectively. Specifically, on the CIFAR-10 dataset, MobileNetV2-FLE-[0.1,0.2,0.3], MobileNetV2-FLE-[0.3,0.6,1.0], ResNet18-FLE-[0.1,0.2,0.3], ResNet18-FLE-[0.3,0.6,1.0] are able to improve 6.83%, 1.84%, 0.43%, and 2.67% accuracy compared to the their primitive DNN models with the same maximum width, respectively. Similarly, on the Fashion-MNIST dataset, consistent results can be observed, as shown in Fig. 4(c)–(d).

Fig. 4(e)–(h) and (i)–(l) exhibit the advantages in computing cost and parameters of MobileNetV2- and ResNet18-derived FLENN models on two datasets, respectively. Similar to [17], we adopt the common multiple-and-accumulate operations (MACs) to quantify computing cost. Obviously, for the MobileNetV2- and ResNet18-derived FLENN models with multiple subnets of different widths, the computing cost and parameters of the subnet increase with width. With respect to computing cost, our MobileNetV2- and ResNet18-derived FLENN models are competitive. For instance, compared to the ResNet18 with the same maximum width on the CIFAR-10 dataset, ResNet18-FLE-[0.1,0.2,0.3] and ResNet18-FLE-[0.3,0.6,1.0] can reduce the computing cost by 46.06% and 53.10%, respectively. Besides, the model parameters of our MobileNetV2- and ResNet18-derived FLENN models can be significantly reduced. For instance, compared to the MobileNetV2 and ResNet18 with the same maximum width

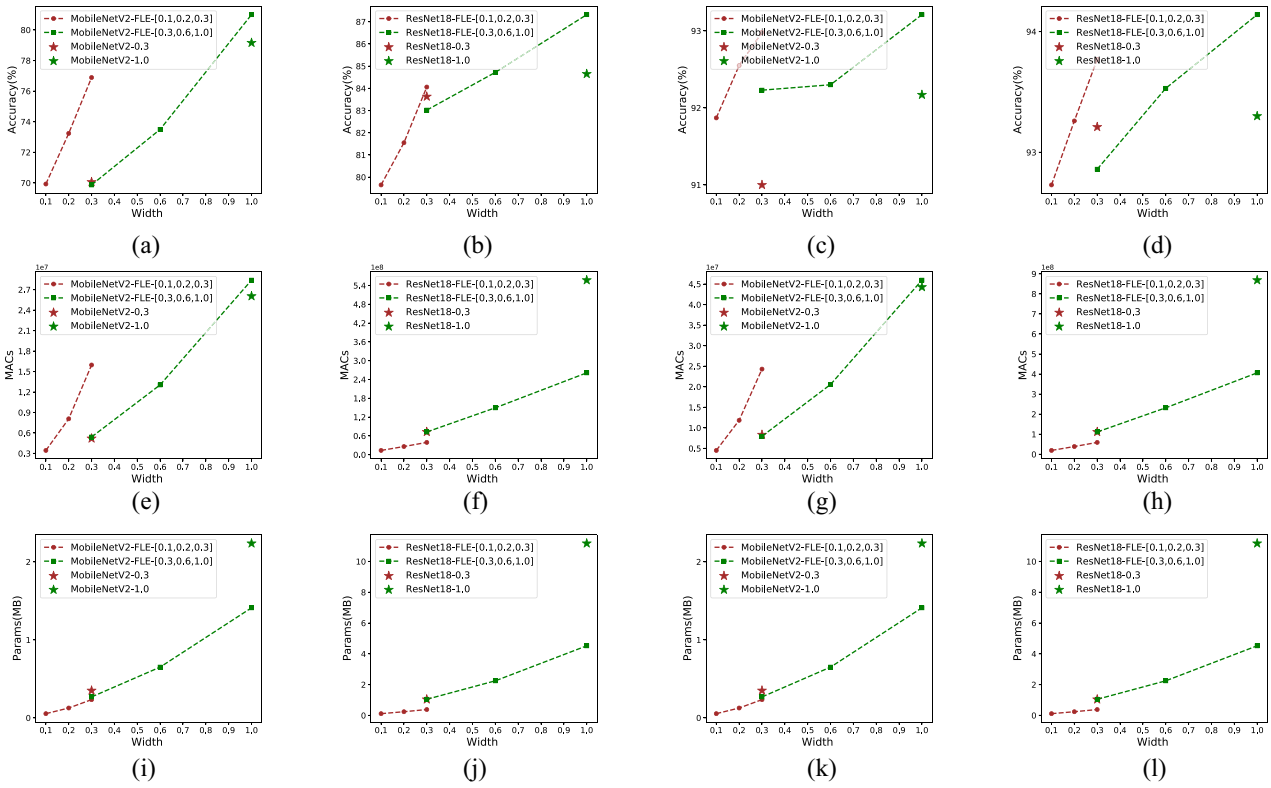


Fig. 4. Performance evaluation of our designed FLENN structure in terms of accuracy, computing cost, and model parameters. (a) Accuracy of MobileNetV2-derived FLENN models versus MobileNetV2 on CIFAR-10. (b) Accuracy of ResNet18-derived FLENN models versus ResNet18 on CIFAR-10. (c) Accuracy of MobileNetV2-derived FLENN models versus MobileNetV2 on Fashion-MNIST. (d) Accuracy of ResNet18-derived FLENN models versus ResNet18 on Fashion-MNIST. (e) Computing cost of MobileNetV2-derived FLENN models versus MobileNetV2 on CIFAR-10. (f) Computing cost of ResNet18-derived FLENN models versus ResNet18 on CIFAR-10. (g) Computing cost of MobileNetV2-derived FLENN models versus MobileNetV2 on Fashion-MNIST. (h) Computing cost of ResNet18-derived FLENN models versus ResNet18 on Fashion-MNIST. (i) Parameters of MobileNetV2-derived FLENN models versus MobileNetV2 on CIFAR-10. (j) Parameters of ResNet18-derived FLENN models versus ResNet18 on CIFAR-10. (k) Parameters of MobileNetV2-derived FLENN models versus MobileNetV2 on Fashion-MNIST. (l) Parameters of ResNet18-derived FLENN models versus ResNet18 on Fashion-MNIST.

on the CIFAR-10 dataset, MobileNetV2-FLE-[0.1,0.2,0.3], MobileNetV2-FLE-[0.3,0.6,1.0], ResNet18-FLE-[0.1,0.2,0.3], and ResNet18-FLE-[0.3,0.6,1.0] can reduce the number of parameters by 33.33%, 36.92%, 66.32%, and 59.54%, respectively. The advantages in parameters and computing cost of our FLENN structure are due to the fact that the unique branch module design of the subnets can greatly reduce the number of neurons and computing cost in convolutional layers of branches, please refer to Fig. 1.

C. Performance Evaluation of Our Freezing-Assisted Wide-to-Narrow Training Mechanism

To demonstrate the effectiveness of our designed freezing-assisted wide-to-narrow training mechanism, this section adopts MobileNetV2- and ResNet18-derived FLENN models of widths [0.1,0.2,0.3] and [0.3,0.6,1.0] for accuracy evaluation on CIFAR-10 and Fashion-MNIST datasets.

In this work, two state-of-the-art training mechanisms for dynamic neural networks are employed as baselines. That is,

1) *Synchronous training mechanism (Synch)* [34] trains all subnets in a dynamic DNN model in the narrow-to-wide fashion, and this Synch updates all parameters of the current subnet.
2) *Stepwise incremental training mechanism (SI)* [8] trains all the subnets in a dynamic DNN model in the narrow-to-wide fashion, and this SI only updates incremental parameters of the current subnet compared to the previous narrower subnet.

TABLE II
ACCURACY COMPARISON OF MOBILENETV2- AND RESNET18-DERIVED FLENN MODELS WITH WIDTHS OF [0.1,0.2,0.3] UNDER DIFFERENT TRAINING MECHANISMS ON CIFAR-10

Model	Training	Widths w			Avg Acc
		0.1	0.2	0.3	
MobileNetV2-FLE	SI [8]	68.85%	71.29%	71.95%	70.70%
	Synch [34]	47.01%	47.69%	64.36%	53.02%
	Proposed	67.25%	71.93%	76.51%	**71.89%(+18.88%)**
ResNet18-FLE	SI [8]	79.54%	80.64%	80.67%	80.28%
	Synch [34]	73.06%	73.02%	73.15%	73.08%
	Proposed	80.24%	81.5%	83.92%	**81.89%(+8.81%)**

Tables II, III, IV, and V present the accuracy of different training mechanisms for FLENN models of widths [0.1,0.2,0.3] and [0.3,0.6,1.0] on two datasets. Clearly, for MobileNetV2- and ResNet18-derived FLENN models of widths [0.1,0.2,0.3] and [0.3,0.6,1.0] on the CIFAR-10 dataset, compared to the Synch and SI training mechanisms, our proposed freezing-assisted wide-to-narrow training mechanism can improve the average training accuracy by 18.88%, 8.81%, 25.52%, and 1.94%, respectively. Meanwhile, the results show that our training mechanism can obtain similar results on the Fashion-MNIST dataset. In addition, we also evaluate the

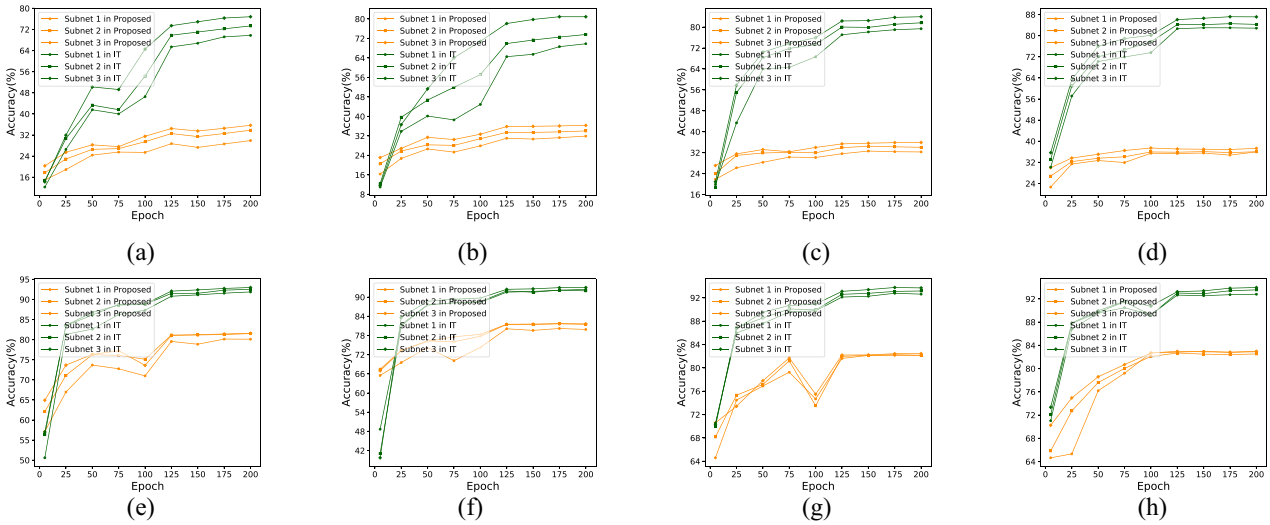


Fig. 5. Training accuracy comparison of MobileNetV2- and ResNet18-derived FLENN models in EFLCF and IT.

TABLE III
ACCURACY COMPARISON OF MOBILENETV2- AND RESNET18-DERIVED FLENN MODELS WITH WIDTHS OF [0.3,0.6,1.0] UNDER DIFFERENT TRAINING MECHANISMS ON CIFAR-10

Model	Training	Widths w			Avg Acc
		0.3	0.6	1.0	
MobileNetV2-FLE	SI [8]	74.39%	74.85%	73.94%	74.39%
	Synch [34]	43.37%	43.8%	62.4%	49.86%
	Proposed	72.07%	73.89%	80.15%	**75.37%(+25.52%)**
ResNet18-FLE	SI [8]	84.7%	85.11%	84.6%	84.81%
	Synch [34]	82.76%	82.93%	83.17%	82.95%
	Proposed	83.35%	84.23%	87.1%	**84.89%(+1.94%)**

TABLE IV
ACCURACY COMPARISON OF MOBILENETV2- AND RESNET18-DERIVED FLENN MODELS WITH WIDTHS OF [0.1,0.2,0.3] UNDER DIFFERENT TRAINING MECHANISMS ON FASHION-MNIST

Model	Training	Widths w			Avg Acc
		0.1	0.2	0.3	
MobileNetV2-FLE	SI [8]	91.46%	91.56%	91.36%	91.46%
	Synch [34]	91.02%	91.08%	90.95%	91.02%
	Proposed	91.61%	92.13%	92.64%	**92.13%(+1.11%)**
ResNet18-FLE	SI [8]	92.66%	92.88%	92.64%	92.73%
	Synch [34]	92.33%	92.35%	92.25%	92.31%
	Proposed	92.62%	92.83%	93.48%	**92.98%(+0.67%)**

accuracy improvement for each subnet of FLENN models. For instance, as can be observed in Table II, compared to SI and Synch, our designed training mechanism can obtain up to 20.24%, 24.24%, and 12.15% accuracy improvements for three subnets of MobileNetV2-FLE-[0.1,0.2,0.3] on CIFAR-10 dataset, in which the average accuracy improvement refers to 18.88%. The training accuracy progress benefits from the wide-to-narrow training technique and the freezing technique. First, the wide-to-narrow training technique can offer a positive pretraining function for narrow subnets. Second, the freezing technique with one random skip manner can provide a helpful regularization function to relieve excess update for these subnets that share an elastic branch module, so as to improve the global training accuracy of FLENN models.

TABLE V
ACCURACY COMPARISON OF MOBILENETV2- AND RESNET18-DERIVED FLENN MODELS WITH WIDTHS OF [0.3,0.6,1.0] UNDER DIFFERENT TRAINING MECHANISMS ON FASHION-MNIST

Model	Training	Widths w			Avg Acc
		0.3	0.6	1.0	
MobileNetV2-FLE	SI [8]	92.14%	92.16%	91.88%	92.06%
	Synch [34]	91.3%	91.18%	90.86%	91.11%
	Proposed	92.14%	92.29%	93.01%	**92.48%(+1.37%)**
ResNet18-FLE	SI [8]	93.63%	93.55%	93.19%	93.46%
	Synch [34]	93.37%	93.22%	93.19%	93.26%
	Proposed	93.25%	93.24%	93.93%	**93.47%(+0.21%)**

D. Necessity of Distributed Collaborative Training

In this section, we conduct extensive experiments to demonstrate the necessity of distributed collaborative training in our proposed EFLCF. Note that in the proposed EFLCF, the number of clients is set to be 100. This work employs a common benchmark IT [39] in which a single client trains its DNN model individually for comparison. As the previous experiments, we adopt the MobileNetV2-FLE-[0.1,0.2,0.3], MobileNetV2-FLE-[0.3,0.6,1.0], ResNet18-FLE-[0.1,0.2,0.3], and ResNet18-FLE-[0.3,0.6,1.0] models with the FLENN structure to evaluate the training accuracy, and record the training accuracy every 25 epochs on the CIFAR-10 and Fashion-MNIST datasets.

Fig. 5(a)-(h) records the accuracy of MobileNetV2-FLE-[0.1,0.2,0.3], MobileNetV2-FLE-[0.3,0.6,1.0], ResNet18-FLE-[0.1,0.2,0.3], and ResNet18-FLE-[0.3,0.6,1.0] models on the CIFAR-10 and Fashion-MNIST datasets during training. Clearly, as demonstrated by the accuracy curves of models, our proposed framework yields remarkable accuracy improvements compared to the IT framework. More specifically, the obtained average accuracy improvements in Fig. 5(a)-(h) are 40.16%, 40.69%, 47.59%, 48.3%, 11.40%, 11.47%, 10.93%, and 10.63%, respectively. This is because our proposed EFLCF with the distributed collaborative training manner can well train FLENN models by exploiting adequate training data across decentralized clients.

TABLE VI
COMPUTING COST EVALUATION

Framework	Computing cost of clients	Computing cost of server	Training accuracy
Traditional FL	80.69×10^6 MACs	3.82×10^6 MACs	81.56%
Proposed without Freezing and Random Skip	1.9×10^6 MACs	82.61×10^6 MACs	81.56%
Proposed	$\mathbf{0.62 \times 10^6}$ **MACs**	$\mathbf{69.61 \times 10^6}$ **MACs**	**81.75%**

E. Computing Cost Evaluation

In this section, we compare our proposed EFLCF with two FL frameworks to validate the advantages of computing cost reduction on IoT devices. In addition, we employ the ResNet18-FLE-[0.1,0.2,0.3] model on the CIFAR-10 dataset for evaluation and analysis. The two adopted benchmarks are summarized as follows.

1) Benchmark 1 is the traditional FL framework [14], in which clients perform forward propagation and backward propagation locally and then upload the local models to a server for aggregation.
2) Benchmark 2 is our proposed EFLCF framework without freezing and random skip strategy, in which the freezing strategy and the random skip strategy in the proposed FL framework are removed.

As listed in Table VI, our proposed EFLCF and two benchmarks are evaluated by three metrics, i.e., computing cost of clients, computing cost of the server, and model accuracy during training. Clearly, the client-side computing cost in our proposed EFLCF is only 2.35% of that of the traditional FL framework. The reason for achieving such a remarkable computing cost reduction on clients is that in our proposed EFLCF, clients offload most of the computing cost to the server, and clients only need to undertake the computing cost of the forward and backward propagation of the head convolution module in ResNet18-FLE-[0.1,0.2,0.3] model. This advantage frees up the training computing limitations of IoT devices and enables more computing-constrained IoT devices to participate in FL training. In addition, compared to Benchmark 2 which does not adopt freezing and random skip strategy, our proposed EFLCF can reduce the computing cost of clients and the server by 67.37% and 15.74%, respectively, while achieving 0.19% accuracy improvement. The above benefits from the facts that the freezing strategy can reduce the updating frequency of the head convolution module on clients, and the random skip strategy skips some updates of intermediate subnets on the server, which can provide a regularization effect.

F. Accuracy and Computing Cost Evaluation of Our Entropy-Based Narrow-to-Wide Elastic Inference Mechanism

To demonstrate the effectiveness of our designed entropy-based narrow-to-wide elastic inference mechanism, this section adopts the well-known MobileNetV2-1.0 model and its derived MobileNetV2-FLE-[0.3,0.6,1.0] model to evaluate the computing cost and accuracy of the inference. Here, MobileNetV2-1.0 refers to a well-trained MobileNetV2 model of width 1.0 and has only one exit, and MobileNetV2-FLE-[0.3,0.6,1.0] refers to the MobileNetV2-derived FLENN model which contains three subnets of widths 0.3, 0.6, and 1.0. Here, for MobileNetV2-FLE-[0.3,0.6,1.0], the confidence levels of the front two subnets are set to [0.18,0.06] and [0.28,0.32].

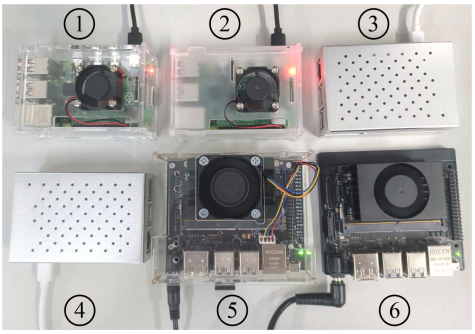


Fig. 6. Real test-bed platform.

Table VII shows the results on accuracy and computing cost comparison during the inference process. As observed, by employing our entropy-based narrow-to-wide elastic inference mechanism, MobileNetV2-FLE-[0.3,0.6,1.0] can significantly reduce the computing cost and improve model accuracy during the inference process. Specifically, compared to MobileNetV2-1.0, our proposed MobileNetV2-FLE-[0.3,0.6,1.0] is able to reduce the computing cost by 42.5% and improve the inference accuracy by 0.34% on average when the confidence level is set to [0.28,0.32]. Moreover, compared to MobileNetV2-1.0, our proposed MobileNetV2-FLE-[0.3,0.6,1.0] is able to reduce the computing cost by 22.5% and improve the inference accuracy by 1.86% on average when the confidence level is set to [0.18,0.06]. The reduction in computing cost is due to the fact that our FLENN model structure allows a lot of easy-to-handle samples to exit from the narrow subnets in a fast manner.

G. Evaluation on Real Test-Bed

To evaluate the effectiveness of our proposed framework in real-world scenarios, we construct a real distributed FL training platform which consists of six heterogeneous client devices and one server, as shown in Fig. 6. Here, the specifications of the heterogeneous devices and the server are listed in Table VIII. In addition, we employ the well-recognized Dirichlet distribution [40] to generate NON-IID data distributions, where the coefficient α controls the degree of data heterogeneity, and the smaller α indicates the higher degree of data heterogeneity. Based on two common NON-IID distributions ($\alpha = 0.3$ and $\alpha = 0.7$) on the Fashion-MNIST dataset, we compare our proposed framework with benchmarks by using the ResNet18-FLE-[0.3,0.6,1.0] model.

Fig. 7 illustrates the latency required by using different methods to achieve the target accuracy under two NON-IID data distributions ($\alpha = 0.3$ and $\alpha = 0.7$), and the detailed numerical results are reported in Table IX. It can be observed that our proposed method achieves the highest accuracy with the lowest latency. Numerically, our proposed method achieves even higher accuracy while delivering speedup gains of up to 6.9× and 14.7× under the two data distributions, respectively. These results demonstrate that our method retains the advantages of accuracy and latency under heterogeneous

TABLE VII
RESULTS ON ACCURACY AND COMPUTING COST COMPARISON DURING INFERENCE

Model	Threshold	Exit Rate / Accuracy of Exit 1	Exit Rate / Accuracy of Exit 2	Exit Rate / Accuracy of Exit 3	Computing cost reduction	Accuracy
MobileNetV2-1.0	✗	✗	✗	✗	26.05×10^6 MACs	79.16%
MobileNetV2-FLE -[0.3,0.6,1.0]	[0.28,0.32]	44.6% / 92.2%	20.7% / 77.9%	34.7% / 64.1%	$\mathbf{14.98 \times 10^6}$ **MACs**	**79.50%**
MobileNetV2-FLE -[0.3,0.6,1.0]	[0.18,0.06]	32.5% / 95.7%	4.9% / 98.9%	62.6% / 71.9%	$\mathbf{20.18 \times 10^6}$ **MACs**	**81.02%**

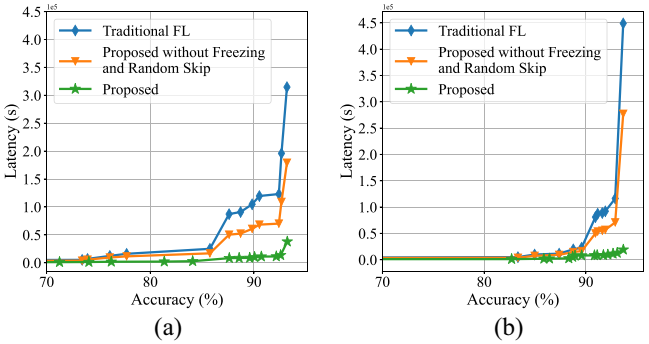


Fig. 7. Compare latency required to achieve target accuracy. (a) NON-IID $\alpha = 0.3$. (b) NON-IID $\alpha = 0.7$.

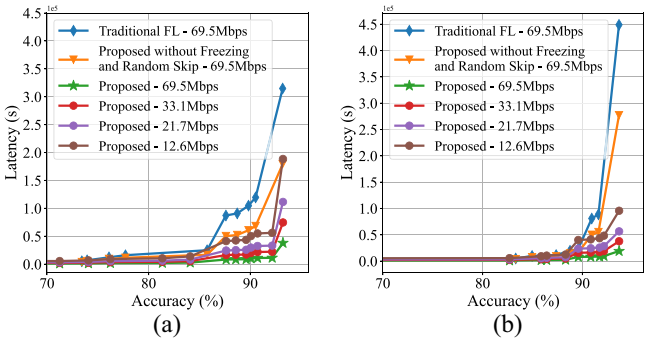


Fig. 8. Compare latency required to achieve target accuracy over different network bandwidth. (a) NON-IID $\alpha = 0.3$. (b) NON-IID $\alpha = 0.7$.

TABLE VIII
CONFIGURATIONS OF THE REAL TEST-BED PLATFORM

ID	Device	Computing Resource	Memory
Server	Workstation	NVIDIA GeForce RTX 4090 GPU	64GB
Client#1	Raspberry Pi 4B	1.5GHz Cortex-A72 ARM CPU	8GB
Client#2	Raspberry Pi 4B	1.5GHz Cortex-A72 ARM CPU	8GB
Client#3	Raspberry Pi 5	2.4GHz Cortex-A76 ARM CPU	8GB
Client#4	Raspberry Pi 5	2.4GHz Cortex-A76 ARM CPU	8GB
Client#5	Jetson Nano B01	128-core NVIDIA Maxwell GPU	4GB
Client#6	Jetson TX2 NX	256-core NVIDIA Pascal GPU	4GB

TABLE IX
REQUIRED LATENCY TO ACHIEVE TARGET ACCURACY

Data distribution	Framework	Accuracy	Latency (s)
NON-IID $\alpha = 0.3$	Traditional FL	93.18%	3.15×10^5
	Proposed without Freezing and Random Skip	93.18%	1.80×10^5
	Proposed	**93.29%**	$\mathbf{0.46 \times 10^5}$
NON-IID $\alpha = 0.7$	Traditional FL	93.67%	4.49×10^5
	Proposed without Freezing and Random Skip	93.67%	2.77×10^5
	Proposed	**93.93%**	$\mathbf{0.30 \times 10^5}$

TABLE X
MEMORY USAGE AND COMPUTATIONAL ENERGY CONSUMPTION

Framework	Memory Usage	Energy Consumption of clients	Energy Consumption of server
Traditional FL	110.02 MB	1124.95 KJ	0.66 KJ
Proposed without Freezing and Random Skip	76.20 MB	4.79 KJ	2280.63 KJ
Proposed	76.20 MB	0.28 KJ	558.29 KJ

data distributions in the realistic hardware platform composed of heterogeneous IoT devices.

Table X reports the results of the memory usage and computational energy consumption for both the clients and server by using different methods. Here, the computational energy consumption is obtained by the measurement tool in [15]. As observed, our proposed method presents significant advantages in terms of the memory usage and energy efficiency. In terms of the memory usage, compared to traditional FL, both variants of our proposed method can reduce the memory usage by 30.7%. This is because our proposed framework only deploys the head convolution module of the FLENN model on the client side, and this module contains few parameters. In addition, compared to the benchmarks, our approach can reduce the total computational energy consumption by up to 50.4%. The reason for the above fact is that the freezing and skipping mechanism can effectively reduce the overall computational workload and avoid the corresponding energy consumption overhead.

H. Communications Overhead Evaluation

Fig. 8 compares the latency of achieving the target accuracy under different bandwidth constraints. Here, under two common NON-IID data distribution scenarios ($\alpha = 0.3$ and $\alpha = 0.7$), we employ the ResNet18-FLE-[0.3,0.6,1.0] model on the Fashion-MNIST dataset for evaluation. It is clearly seen from the figure that, compared with benchmarks, our method is able to achieve the target accuracy with lower latency under various limited bandwidth 69.5, 33.1, 21.7, and 12.6 Mb/s conditions. For instance, as shown in Fig. 8(b), in the NON-IID data distribution scenario ($\alpha = 0.7$), compared to the conventional FL with the highest bandwidth of 69.5 Mb/s, our scheme can achieve 95.8%, 91.6%, 87.4%, and 78.7% reductions in training latency respectively under same bandwidth 69.5 Mb/s, and limited bandwidth 33.1, 21.7, and 12.6 Mb/s conditions. The reason for the above latency improvement is that our scheme can well alleviate the computing cost pressure of computing-constrained devices.

TABLE XI
COMMUNICATION OVERHEAD OF ONE TRAINING BATCH

Framework	Communication Overhead
Proposed without Freezing and Random Skip	55.17 MB
Proposed	**7.36 MB**

In addition to the latency evaluation under different bandwidth conditions, Table XI elaborates on the improvement in communication overhead of our designed freezing-assisted wide-to-narrow training mechanism. Clearly, it can be observed that communication overhead is reduced by 86.7% in each training batch. The above reduction in communication overhead is attributed to the fact that our randomized skip strategy and freezing-assisted technique can significantly reduce the frequency of transmitting the input tensor from the client to the server as well as the frequency of returning the gradient from the server to the client. Thus, it can be concluded that our approach can mitigate the communication overhead well.

VIII. CONCLUSION

To alleviate the high computing cost of IoT devices in emerging FL, we propose in this article an elastic FL collaboration framework via coupling an FLENN model with multiple-width subnets. The supportive training and inference mechanisms are proposed to exploit the advantages of our proposed framework. More specifically, a freezing-assisted wide-to-narrow training mechanism is designed to efficiently accomplish device-server collaborative training with low device computing cost, and an entropy-based narrow-to-wide elastic inference mechanism to optimize computing cost without degrading accuracy. Experimental results demonstrate that our approach can significantly improve the distributed training accuracy as well as the device computing cost in FL training and inference.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE/ACM Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[2] Z. Ma, J. Pan, and J. Chai, "World-to-words: Grounded open vocabulary acquisition through fast mapping in vision-language models," in *Proc. Annu. Meeting Assoc. Comput. Linguist.*, 2023, pp. 524–544.

[3] A. Dhavlle, M. M. Ahmed, N. Mansoor, K. Basu, A. Ganguly, and S. M. P. Dinakarrao, "Defense against on-chip trojans enabling traffic analysis attacks based on machine learning and data augmentation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 12, pp. 4681–4694, Dec. 2023.

[4] C. Wang, G. Yang, G. Papanastasiou, H. Zhang, J. J. P. C. Rodrigues, and V. Albuquerque, "Industrial cyber-physical systems-based cloud IoT edge for federated heterogeneous distillation," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5511–5521, Aug. 2021.

[5] L. Ren, T. Wang, Z. Jia, F. Li, and H. Han, "A lightweight and adaptive knowledge distillation framework for remaining useful life prediction," *IEEE Trans. Ind. Informat.*, vol. 19, no. 8, pp. 9060–9070, Aug. 2023.

[6] W. Li, A. Hu, N. Xu, and G. He, "A precision-scalable deep neural network accelerator with activation sparsity exploitation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 1, pp. 263–276, Jan. 2024.

[7] Y. Ping, H. Jiang, X. Liu, Z. Zhao, Z. Zhou, and X. Chen, "Latency-based inter-operator scheduling for CNN inference acceleration on GPU," *IEEE Trans. Services Comput.*, vol. 17, no. 1, pp. 277–290, Jan./Feb. 2024.

[8] H. Wang, L. Li, Y. Cui, N. Wang, F. Shen, and T. Wei, "MBSNN: A multi-branch scalable neural network for resource-constrained IoT devices," *J. Syst. Archit.*, vol. 142, Sep. 2023, Art. no. 102931.

[9] K. Cao, S. Hu, Y. Shi, A. W. Colombo, S. Karnouskos, and X. Li, "A survey on edge and edge-cloud computing assisted cyber-physical systems," *IEEE Trans. Ind. Informat.*, vol. 17, no. 11, pp. 7806–7819, Nov. 2021.

[10] D. Wu, W. Yang, H. Jin, X. Zou, W. Xia, and B. Fang, "FedComp: A federated learning compression framework for resource-constrained edge computing devices," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 1, pp. 230–243, Jan. 2024.

[11] Y. Cui, K. Cao, G. Cao, M. Qiu, and T. Wei, "Client scheduling and resource management for efficient training in heterogeneous IoT-edge federated learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 8, pp. 2407–2420, Aug. 2022.

[12] M. Chen, N. Shlezinger, H. V. Poor, Y. C. Eldar, and S. Cui, "Communication-efficient federated learning," *Proc. Nat. Academy Sci.*, vol. 118, no. 17, 2021, Art. no. e2024789118.

[13] X. Zhou, C. Liu, and J. Zhao, "Resource allocation of federated learning for the metaverse with mobile augmented reality," *IEEE Trans. Wireless Commun.*, early access, Oct. 31, 2023, doi: 10.1109/TWC.2023.3326884.

[14] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Stat.*, 2017, pp. 1273–1282.

[15] Y. Cui, K. Cao, J. Zhou, and T. Wei, "Optimizing training efficiency and cost of hierarchical federated learning in heterogeneous mobile-edge cloud computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 5, pp. 1518–1531, May 2023.

[16] A. Ignatov et al., "AI benchmark: Running deep neural networks on android smartphones," in *Proc. Eur. Conf. Comput. Vis.*, 2019, pp. 288–314.

[17] Y. Cui, Z. Zhang, N. Wang, L. Li, C. Chang, and T. Wei, "User-distribution-aware federated learning for efficient communication and fast inference," *IEEE Trans. Comput.*, vol. 73, no. 4, pp. 1004–1018, Apr. 2024.

[18] A. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[19] J. Zhao, L. Yang, B. Zhang, G. Guo, and D. Doermann, "Uncertainty-aware binary neural networks," in *Proc. Int. Joint Conf. Artif. Intell.*, 2021, pp. 3441–3447.

[20] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–15.

[21] A. Eshratifar, M. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 565–576, Feb. 2021.

[22] Y. Xu, T. Mohammed, M. Francesco, and C. Fischione, "Distributed assignment with load balancing for DNN inference at the edge," *IEEE Internet Things J.*, vol. 10, no. 2, pp. 1053–1065, Jan. 2023.

[23] S. Teerapittayanon, B. McDanel, and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Proc. IEEE Int. Conf. Pattern Recognit.*, 2016, pp. 2464–2469.

[24] W. Pang, X. Jiang, M. Lv, T. Gao, D. Liu, and W. Yi, "Toward the predictability of dynamic real-time DNN inference," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 9, pp. 2849–2862, Sep. 2022.

[25] N. Shlezinger, M. Chen, Y. C. Eldar, H. V. Poor, and S. Cui, "UVeQFed: Universal vector quantization for federated learning," *IEEE Trans. Signal Process.*, vol. 69, pp. 500–514, Dec. 2020.

[26] L. Li, D. Shi, R. Hou, H. Li, M. Pan, and Z. Han, "To talk or to work: Flexible communication compression for energy efficient federated learning over heterogeneous mobile edge devices," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2021, pp. 1–10.

[27] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 10, pp. 4229–4238, Oct. 2020.

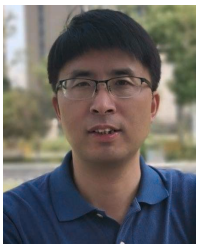
[28] Y. Zhan, P. Li, and S. Guo, "Experience-driven computational resource allocation of federated learning by deep reinforcement learning," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2020, pp. 234–243.

[29] N. H. Tran, W. Bao, A. Zomaya, M. N. Nguyen, and C. S. Hong, "Federated learning over wireless networks: optimization model design and analysis," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2019, pp. 1387–1395.

[30] G. Wang, F. Xu, H. Zhang, and C. Zhao, "Joint resource management for mobility supported federated learning in Internet of Vehicles," *Future Gener. Comput. Syst.*, vol. 129, pp. 199–211, Apr. 2022.
[31] J. Yu and T. Huang, "Universally slimmable networks and improved training techniques," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1803–1811.
[32] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[33] M. Tan and Q. V. Le, "EfficientNetV2: Smaller models and faster training," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 10096–10106.
[34] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–12.
[35] G. Huang, D. Chen, T. Li, F. Wu, L. Maaten, and K. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–14.
[36] Y. Cui, L. Li, Z. Tao, M. Chen, and T. Wei, "Filtering out high noise data for distributed deep neural networks," *IEEE Trans. Autom. Sci. Eng.*, vol. 20, no. 1, pp. 101–111, Jan. 2023.
[37] T. Zhou, J. Zhang, and D. H. K. Tsang, "FedFA: Federated learning with feature anchors to align features and classifiers for heterogeneous data," *IEEE Trans. Mobile Comput.*, vol. 23, no. 6, pp. 6731–6742, Jun. 2024.
[38] J. Zhang, L. Zhao, K. Yu, G. Min, A. Y. Al-Dubai, and A. Y. Zomaya, "A novel federated learning scheme for generative adversarial networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 3633–3649, May 2024.
[39] J.-H. Ahn, O. Simeone, and J. Kang, "Wireless federated distillation for distributed edge learning with heterogeneous data," in *Proc. IEEE Int. Annu. Int. Symp. Pers., Indoor Mobile Radio Commun.*, 2019, pp. 1–6.
[40] Z. Xia et al., "CaBaFL: Asynchronous federated learning via hierarchical cache and feature balance," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 11, pp. 4057–4068, Nov. 2024.

Haizhou Wang received the M.S. degree from the School of Computer Science and Technology, East China Normal University, Shanghai, China, in 2023.

His current research interests are in the areas of Internet of Things, federated learning, and dynamic neural networks.



Guobing Zou received the Ph.D. degree in computer science from Tongji University, Shanghai, China, 2012.

He is a Full Professor and the Vice Dean of the School of Computer Science, Shanghai University, Shanghai. He has worked as a Visiting Scholar with the Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO, USA, from 2009 to 2011. He has published more than 120 papers on premier international journals and conferences. His current research interests mainly focus on services computing, edge computing, and intelligent algorithms.

Kun Cao (Member, IEEE) received the Ph.D. degree in computer science from East China Normal University, Shanghai, China, in 2020.

He is currently an Associate Professor with the College of Cyber Security, Jinan University, Guangzhou, China. His current research interests are in the areas of Internet of things and cyber–physical systems.

Dr. Cao has been serving as an Associate Editor for the *Journal of Circuits, Systems, and Computers* since 2020/10.

Yangguang Cui (Member, IEEE) received the Ph.D. degree in computer science from East China Normal University, Shanghai, China, in 2023.

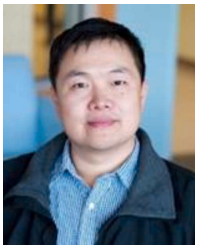
He is currently an Assistant Professor with the School of Computer Engineering and Science, Shanghai University, Shanghai. His current research interests are in the areas of Internet of Things and federated learning.

Dr. Cui has been serving as an Associate Editor for the *Journal of Circuits, Systems, and Computers*, since 2023/7.

Tongquan Wei received the Ph.D. degree in computer engineering from Michigan Technological University, Houghton, MI, USA, in 2009.

He is a Professor with the School of Computer Science and Technology, East China Normal University, Shanghai, China. He has published more than 100 refereed papers. His research interests are in the context of edge computing and cyber physical systems.

Prof. Wei is currently an Associate Editor of the IEEE TRANSACTIONS ON SERVICES COMPUTING.



Shiyan Hu received the Ph.D. degree in computer engineering from Texas A&M University, College Station, TX, USA, in 2008.

He is a Global STEM Professor with the Department of Data and Systems Engineering, University of Hong Kong, Hong Kong. His research interests include cyber–physical systems, cyber–physical system security, and data analytics, where he has published more than 180 refereed papers, including about 100 in IEEE Transactions.