FedSNN: Training Slimmable Neural Network with Federated Learning in Edge Computing

Yang Xu, *Member, IEEE,* Yunming Liao, Hongli Xu, *Member, IEEE,* Zhiyuan Wang, Lun Wang, Jianchun Liu, Chen Qian, *Senior Member, IEEE*

Abstract—To provide a flexible tradeoff between inference accuracy and resource requirement at runtime, the slimmable neural network (SNN), a single network executable at different widths with the same deploying and management cost as that of a single model, has been proposed. However, how to effectively train SNN among massive devices in edge computing without revealing their local data remains an open problem. To this end, we leverage a novel distributed machine learning paradigm, *i.e.*, federated learning, to realize effective on-device SNN training. As current FL schemes often train only one model with fixed architecture, and the existing SNN training algorithm is resource-intensive, integrating FL and SNN is non-trivial. Furthermore, two intrinsic features in edge computing, *i.e.*, data and system heterogeneity, exacerbate the difficulty. Motivated by this, we redesign the model distribution, local training, and model aggregation phases in traditional FL, and propose FedSNN, a framework that ensures all widths in SNN can obtain high accuracy with less resource consumption. Specifically, for devices with heterogeneous training capacities and data distributions, the parameter server will distribute each of them with one proper width for adaptive local training guided by their uploaded model features, and their trained models will be weighted-averaged using the proposed multi-width SNN aggregation to improve their statistical utility. Extensive experiments on a distributed testbed show that FedSNN improves the model accuracy by about 2.18%-8.1%, and accelerates training by about 1.31×-6.84×, compared with existing solutions.

Index Terms—*Edge computing, Slimmable neural network, Federated learning, Data and system heterogeneity*

1 Introduction

Recently, increasing amount of data play an important role in the success of deep neural networks (DNNs) [1], [2], which have been widely used in many AI applications, such as image classification and human activity recognition [3]. Nevertheless, since DNNs usually contain a large number of parameters (*e.g.*, 10^9-10^{12} [4], [5]), training and inference using a DNN always impose heavy computation and memory burden [6]–[8]. Hence DNNs on massive and relatively cheap devices in edge computing [9] can hardly satisfy the QoS (*e.g.*, latency or accuracy) of their applications. Specifically, for an image recognition module, it is obvious that online payment applications require a larger model to ensure the recognition accuracy, while a smaller model may be deployed in freeway monitors to reduce inference delay so as to adapt to heavy traffic flow. Different applications pose different QoS requirements and need different models to accommodate such requirements [10], [11]. Moreover, even for one application, the requirement will vary with devices and time, *e.g.*, powerful devices prefer larger models, whereas those low-energy devices can adopt smaller models to reduce resource consumption. To handle these issues, current solutions [12]–[14] often deploy many DNNs for one application on each device and select one proper model at a time to satisfy its requirements.

However, the aforementioned solutions are inflexible and have some constraints, including extra resource consumption for training and cost for storing and managing models with different sizes. Slimmable neural networks (SNNs) [15]–[19] have been proposed with the promise of enabling a single DNN executable at different widths, permitting instant and adaptive accuracy-latency trade-off at runtime. One width can be denoted as a single sub-model, *e.g.*, half-width on each convolutional layer represents that its number of channels is scaled by 0.5. Some typical SNN training algorithms [15], [16], [20] always iteratively and sequentially train each width in each epoch, and then accumulate their gradients for parameter updating [15], [16], [20]. When an SNN is trained and deployed, devices will switch the model size by adopting different width configurations to meet the application requirements and resource constraints.

Traditionally, SNN is trained in a central cloud by collecting all raw data from its generators (*e.g.*, edge devices) and the trained model is then distributed to the devices for inference [15]. However, considering the massive communication cost for moving data to a cloud server as well as the privacy concern associated with raw data sharing, such centralized training raises increasing concerns [21]. Instead, federated learning (FL), which enables devices to collaboratively build a shared model while preserving local data privacy [22]–[24], is a better solution for solving the communication cost and privacy problems. Specifically, an

- *Y. Xu, Y. Liao, H. Xu, Z. Wang, J. Liu and L. Wang are with University of Science and Technology of China, Hefei, Anhui, China, 230027, and also with Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu, China, 215123. E-mail: xuyangcs@ustc.edu.cn, ymliao98@mail.ustc.edu.cn, xuhongli@ustc.edu.cn, cswangzy@mail.ustc.edu.cn, jcliu17@ustc.edu.cn, and wanglun0@mail.ustc.edu.cn.*
- *C. Qian is with the Department of Computer Science and Engineering, Jack Baskin School of Engineering, University of California Santa Cruz, Santa Cruz, CA 95064 USA. E-mail: cqian12@ucsc.edu.*
- *H. Xu is the corresponding author.*

FL system always comprises two components, *i.e.*, the parameter server and devices. Such two components perform model training with three phases. 1) Model distribution: the parameter server distributes the up-to-date model to devices. 2) Local training: each device trains models over its local dataset. 3) Model aggregation: the parameter server collects the updated models for generating a new model. Current FL schemes (*e.g.*, FedAvg [22]) can only train and produce one global model with fixed architecture during each training procedure. Once the model's architecture has been modified, its accuracy will drop precipitously and the model must be retrained [25], [26].

However, this training scheme is also ill-suited in edge computing due to the following reasons. *1) System heterogeneity.* The devices in FL are resource constrained and always equipped with extremely different computing and communication resources [23], [27], [28]. Thus, training multiple widths simultaneously on heterogeneous devices will inevitably cause system performance degradation and bring significant synchronization cost to the entire system. Meanwhile, their memory burden of model training will also increase sharply for storing multiple widths and their gradients. *2) Data heterogeneity.* The data distribution across devices is often non-independently and identically distributed (non-iid) due to devices' different local environments. As a result, the local models of different devices can converge in extremely different directions, which is a major bottleneck to hinder the development of FL [29]–[31]. Training SNN with FL will make such problem even harder. Since SNN consists of many different width configurations, training each width will inevitably interfere with the others. Coupled with the different data distributions among devices, the convergence of global model becomes questionable due to such optimization inconsistency. Hence it is challenging to train SNN with FL while effectively handling system and data heterogeneity in edge computing.

In this paper, we design a novel training framework, namely FedSNN. The fundamental difference from the aforementioned training schemes is that FedSNN redesigns the main training phases of FL, to adapt to the characteristics of SNN and make sure its all widths can be effectively trained, without enforcing each device to train multiple widths in each epoch. Specifically, in FedSNN, for devices with heterogeneous training capacities and data distributions, the parameter server will distribute each of them with one appropriate width configuration for adaptive training, and their trained models will be weighted-averaged using the multi-width SNN aggregation approach to improve their statistical utility.

However, the design of FedSNN involves the following key technical challenges, which will be further discussed in Section 2. *1)* Each width in SNN requires different resources in per-epoch training, and such condition will be exacerbated due to the devices' heterogeneous training capabilities (Figure 1). Thus, the first challenge lies in distributing widths to devices and designing a local training method to alleviate the synchronization barrier caused by system heterogeneity. *2)* Considering data heterogeneity, if different widths are trained over data with different distributions, the convergence of aggregated global SNN becomes questionable (Figure 2). Thus, it is critical to learn devices' data distribution without accessing their private data for model distribution optimization. *3)* Moreover, since devices may train neural networks with different widths on different dataset, we also should optimize model aggregation to learn the data information from devices. Overall, FedSNN demands to determine which width should each device train (model distribution), how many local iterations to undertake on this width (local training) and how to aggregate this width (model aggregation) by considering both the system and data heterogeneity. The main contributions of this paper are summarized as follows:

- We propose a general federated learning framework, called FedSNN, to train width-adjustable SNN by addressing data and system heterogeneity in edge computing. We theoretically prove that FedSNN provides the convergence guarantee for each width during model training.
- For heterogeneous devices, we design novel methods to distribute different widths to the devices for local training, adjust the local iterations for improving their data utility, and optimize the aggregation strategy accordingly to improve their statistical efficiency.
- We conduct extensive experiments on a distributed testbed to show that FedSNN improves model accuracy by about 1.88%-8.1%, accelerates training by about 1.31×-6.84×, and reduces the network traffic consumption by 57.8%-92.2%, compared with existing solutions.

The rest of this paper is organized as follows. Section 2 introduces some background and challenges. Section 3 introduces the system overview of FedSNN. The detailed design is described in Section 4. In Section 5, we provide the theoretical analysis on convergence performance. We report our experimental settings and results in Section 6. We review the related work in Section 7, and conclude the paper in Section 8.

2 Background and Motivation

2.1 Backgrounds on SNN and FL

To achieve a flexible tradeoff between inference accuracy and resource requirement at runtime on edge devices, SNNs [15], [16] have been proposed to make the width of a model become adjustable. Specifically, an SNN has multiple (*e.g.*, M) widths, and each width $m = 1, 2, ..., M$ is denoted as SNN_m for simplicity. Each SNN_m is with a special width-multiplier from 0 to 1, *e.g.*, 0.25×, or 0.5×, and we assume that a larger m denotes a wider width. Let w be the weight vector of the SNN. Since the parameters in the SNN are shared by all widths, each SNN_m can be represented as $w \odot \Gamma_m$, where $\odot$ is the element-wise product and Γ_m is the corresponding binary mask. Physically, these masked parameters will be removed from the model during training. For ease of expression, some important notations in this paper are listed in Table 1.

The emerging paradigm of FL strives to enable devices to cooperatively train models without exposing their raw data. We consider a typical FL system [22], [32] with one parameter server and N end devices. Each device $n = 1, ..., N$ owns local dataset D_n with a distribution as follows,

$$P_n = \{p_n^1, p_n^2, ..., p_n^C\}, \tag{1}$$

TABLE 1: Key Notations.

Notation	Semantics
SNN_m	the SNN model with width m
N	number of devices
D_n	the local dataset of device n
P	the global data distribution
P_n	the data distribution of device n
C	the total number of classes
w_n	the local model on device n
$f_n(w_n)$	the local loss function on device n
ξ	a batch of data samples from D_n
τ_n	the local training iteration of device n

where $C = |P_n|$ is the total number of classes and the proportion p_n^c of the local data samples in D_n is labeled with $c = 1, ..., C$. Similarly, let $P = \{p^1, p^2, ..., p^C\}$ be the global data distribution. Generally, traditional FL consists of three steps. *1) Model distribution.* The parameter server distributes the same global model w to all devices. *2) Local training.* Each device n is associated with a local loss as follows, *i.e.*,

$$f_n(w_n) = \frac{1}{|D_n|} \sum_{\xi \in D_n} F_n(w_n; \xi), \tag{2}$$

where w_n is its local model, ξ is a set of data samples in D_n, $|D_n|$ is the size of local data and $F_n(w_n; \xi)$ is the loss over ξ. To minimize $f_n(w_n)$, device n updates the local model by mini-batch stochastic gradient descent (SGD) [33] for τ iterations as follows,

$$w_n^{\tau'+1} = w_n - \eta \nabla f_n(w_n^{\tau'}), \tag{3}$$

where $0 \leq \tau' < \tau$, η is the learning rate, and $\nabla f_n(w_n^{\tau'})$ is the gradient. *3) Model aggregation.* The parameter server collects models from all devices to generate a new model for the next training epoch. Since the neural network architecture among devices is identical in conventional FL, the aggregation step is performed by model averaging, *i.e.*,

$$w = \frac{1}{N} \sum_n w_n^{\tau}. \tag{4}$$

These three steps will be iteratively performed, until the required termination conditions (*e.g.*, time limit or measured accuracy) are satisfied.

2.2 Challenges for Integrating FL and SNN

In this section, we will experimentally investigate the challenges for integrating FL and SNN in heterogeneous edge computing, and illustrate our motivation.

Impact of System Heterogeneity. Nowadays, many commercial end devices (*e.g.*, smartphones, and NVIDIA TX2/NX) in edge networks are always equipped with different hardware processors [34], [35], which experience significant heterogeneity in floating-point calculation, leading to different computing capacities. Besides, many devices always travel through various types of networks from LANs to WANs, *e.g.*, cellular networks, and the Internet, leading to heterogeneous communication capacities [9], [36]. To investigate the impact of such heterogeneity, we measure the per-epoch overhead of local training (computing) and model aggregation (communication) among different end devices (*i.e.*, TX2 and NX) by taking the default experimental settings in Section 6.

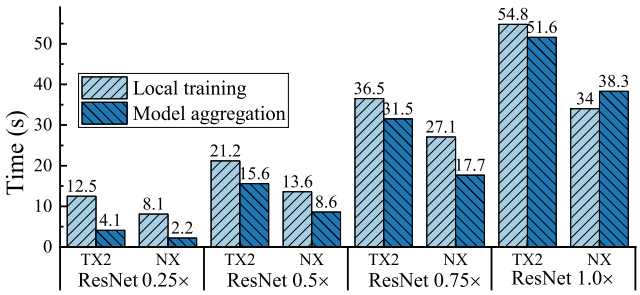


Fig. 1: Per-epoch time duration of local training and model aggregation on different commercial devices.

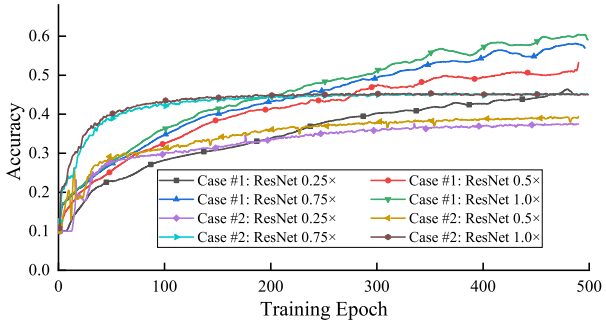


Fig. 2: Convergence performance of different widths with two model distribution approaches.

From the results in Figure 1, we observe that the system heterogeneity spans two aspects. Firstly, an SNN consists of multiple widths, and the resource requirement of each width varies significantly. The wider the model becomes, the more resources it demands. For example, the computing and communication costs of ResNet 1.0× is about 5× and 16× than those of ResNet 0.25×, respectively. Secondly, when training the same width, system performance across devices exhibits significant differences in both computing and communication capacities. For example, when training width 0.5×, the overhead of local training and model aggregation on TX2 is 21.2s and 15.6s, but the corresponding delay on NX is 13.6s and 8.6s, respectively. Thus, training identical width(s) by heterogeneous devices in FedAvg or SlimFL will inevitably cause system performance degradation, and each training epoch only processes as fast as the slowest devices. To alleviate the synchronization barrier during SNN training, the most straightforward solution is to assign weaker devices with smaller widths. However, the operation has little effect on the data heterogeneity, which also influences the training performance to a great extent.

Impact of Data Heterogeneity. Different from centralized training, the local data distributions of different devices will vary heavily, which inevitably incurs the non-iid issue in FL [37]. Specifically, each device's local data distribution is often not a good representative of the global distribution [29], [38], and their local models may easily converge in different directions, resulting in global accuracy degradation [29], [39]. Furthermore, in SNN, this challenge transfers to

two levels: the distribution difference between local data owned by each device, and that between the unions of the data each width is trained on. To observe the impact of non-iid data, we conduct the following set of experiments. We assign the data samples with the first half of the labels to the first half of devices randomly and uniformly, and the remaining samples are distributed to the remaining devices. In case #1, we pick two devices (*i.e.*, one from the first group of devices and one from the second group) to train the same width. In case #2, two devices that train the same width are picked from the same group of devices.

The results in Figure 2 show that each width in case #1 converges faster than that in case #2 at the beginning of the training with a sacrifice of the final model accuracy. Note that in case #2, the devices that train the same width are with the same data distribution. Thus, their local models are optimized at similar direction and their aggregated global model obtains a good performance at the early stage. However, since each width cannot learn the information from the data samples with the remaining half of labels, it converges at low accuracy. Instead, in case #1, for the set of devices that train the same width, the distribution of the union of their local data is closer to global distribution, and thus the accuracy of global models keeps growing even after 500 epochs, and the accuracy improvement exceeds 10%. Overall, although it is impossible to modify or share the devices' local dataset for reducing their distribution difference, we may adopt some model distribution strategies and make each width be trained over the data with similar distribution. In this way, the aggregated models on each width can be optimized to approach the global optima.

Motivation. The above results verify that FedSNN enables great potential in performance improvement (*i.e.*, accuracy or latency). However, two intrinsic features in edge computing (*i.e.*, system and data heterogeneity) have also brought challenges to FedSNN, and limited its effectiveness. To this end, for incorporating FL into SNN more effectively, we should exploit both the data and system heterogeneity, and judiciously optimize all main training phases in conventional FL. Specifically, we can balance the workload of devices by assigning width for each device to adapt to its training ability and let non-stragglers train more data samples to improve the data efficiency. Simultaneously, we should ensure each width is trained over global distribution such that the convergence of aggregated model can be enhanced. The detailed design will be illustrated in the following sections.

3 Overview of FedSNN

In this section, we will introduce the key modules in FedSNN and its training workflow among the parameter server and devices. Figure 3 presents the overall design. Specifically, two modules (*i.e.*, *model aggregator* and *model distributor*) in the parameter server cooperate to coordinate all distributed devices by performing model distribution and aggregation. Besides, each device is deployed with three modules (*i.e.*, *model receiver*, *model updater*, and *model uploader*) to perform the local training on its local data. The detailed training workflow among these modules in each training epoch is as follows:

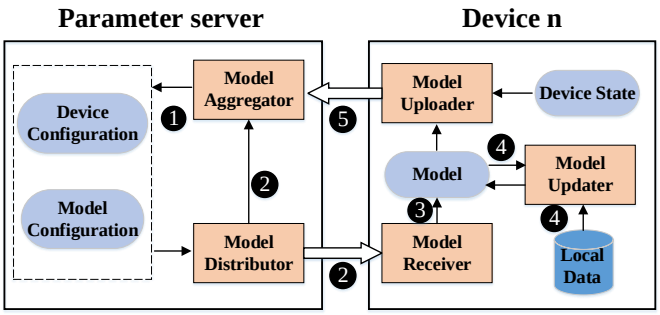


Fig. 3: The training workflow of FedSNN among parameter server and an arbitrary device n.

❶ At the parameter server side, along with the devices' feedback from the last training epoch, the *model aggregator* will perform model aggregation among these collected models to update the global model and the device configuration (*e.g.*, the accessible device, and their available resources), respectively.

❷ To accommodate the system and data heterogeneity in edge computing, the *model distributor* utilizes the up-to-date model and device configurations, and performs model distribution for the next epoch, *i.e.*, determining the width of SNN assigned to each device for local training. After distributing the corresponding profiles to devices, it feeds the distribution results to *model aggregator* to guide the model aggregation.

❸ At the device side, the *model receiver* is responsible for communicating with the server-side *model distributor* to receive model for adaptive local training.

❹ The *model updater* module pulls the stored local model and data samples from parameter store and local dataset, respectively. It then adapts the number of local iterations to perform on the data independently and updates the model parameters, accordingly.

❺ Consequently, the *model uploader* forwards the updated model parameters along with the device state (*e.g.*, network bandwidth, computing resource) to the parameter server, and progresses to the next epoch.

By iteratively executing the aforementioned five steps (❶-❺), each device can eventually obtain a well-trained SNN model. In the following sections, we will primarily focus on three key modules in FedSNN (*i.e.*, model distributor, model updater and model aggregator), and redesign the corresponding training phase (model aggregation, local training and model aggregation) in traditional FL to effectively handle system heterogeneity and data heterogeneity in edge computing when integrating FL into SNN.

4 System Design

In this section, we elaborate the detailed designs of adaptive model distribution, local training optimization and multi-width SNN aggregation modules in FedSNN.

4.1 Adaptive Model Distribution

This section presents the detailed design of model aggregation in FedSNN. To address the system and data

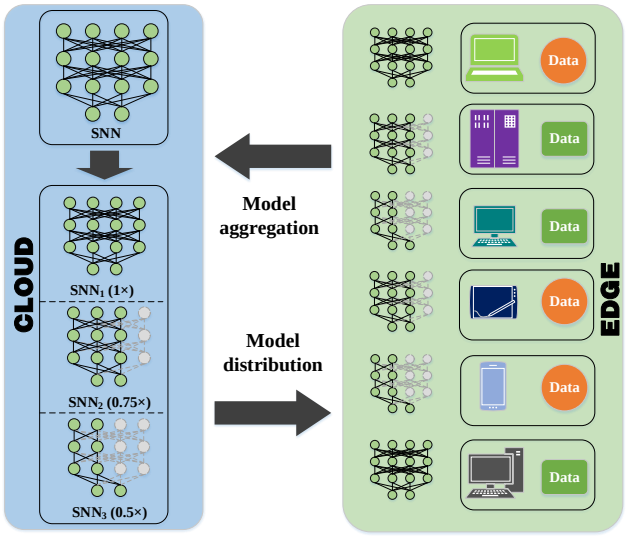


Fig. 4: Adaptive model distribution in FedSNN. Different shapes (circle or rectangle) of devices' data represent their different local data distributions.

heterogeneity, the parameter server in FedSNN adaptively distributes models with different width configurations to various devices for local training, as illustrated in Figure 4. As previously noted, training models with different widths incurs varying computational and communication costs in FL, *i.e.*, the smaller the width is, the less computing and communication resources are required. Thus, to reduce the synchronization cost among devices, we prioritize low-resource devices to train lightweight widths. For example, the smartphone in Figure 4 trains width 0.5× while the computer adopts width 1.0×.

To ensure that all widths reach the required target accuracy, we exploit data heterogeneity by training each SNN_m on a dataset with a distribution approaching the global distribution. Specifically, such distribution is determined by the set of devices that each SNN_m is assigned to, and is defined as $P_m = \{p_m^1, ..., p_m^C\}$, where p_m^c is computed by

$$p_m^c = \frac{\sum_{n=1}^{N} x_{n,m} p_n^c |D_n|}{\sum_{n=1}^{N} x_{n,m} |D_n|}, \tag{5}$$

and $x_{n,m} \in \{0, 1\}$ represents whether SNN_m is assigned to device n or not. Hence, using the earth mover's distance [29], the data distribution difference among P_m and the global distribution P can be calculated as $\sum_{c=1}^{C} ||p_m^c - p^c||$. Given a small positive value ε, we can formulate the constraint $\sum_{c=1}^{C} ||p_m^c - p^c|| \leq \varepsilon$ to achieve similar data distribution for each SNN_m. However, in FL, privacy concerns have ruled out any possibility of the parameter server accessing the raw data on each device [21], [29], and thus FedSNN cannot profile devices' local data distribution directly and justify whether this constraint is satisfied or not. To this end, we design the model parameters based approach to optimize the model distribution without accessing devices' local data distributions. Specifically, there is always an implicit connection between the distribution of local data on one device and the model parameters trained based on these data [29], [32], [40], which enables the parameter server to profile devices' data distribution based on their trained local models in each epoch. Next, we formally present our proposed approach for training:

Algorithm 1 Procedure of device-clustering

1: Select $|\mathbb{S}|$ arbitrary devices;
2: Set the centroid $\mathbb{C}_s^*$ for each cluster s;
3: **repeat**
4: **for** each device n **do**
5: Calculate $d(w_n, \mathbb{C}_s^*)$;
6: **if** $\sum_{n=1}^{N} y_n^s < M$ **then**
7: Assign device n to the cluster s with $\sum_{n=1}^{N} y_n^s < M$;
8: **else**
9: Assign device n to the cluster s with minimum $d(w_n, \mathbb{C}_s^*)$;
10: **end if**
11: **end for**
12: **for** each cluster s **do**
13: Update the centroid as the aggregated model;
14: **end for**
15: **until** the cluster structure does not change or the objective function in Eq. (6) does not decrease anymore;
16: **return** clustering results.

Device Clustering. Upon receiving the updates from devices, the parameter server first performs device-clustering according to their model parameters. The local data distribution influences the model optimization direction on each device [29], [38], and thus we propose to measure the similarity between optimization directions of two devices by inspecting their local models' cosine similarity (CS) [41], [42]. Then, we apply the clustering algorithm so as to minimize the objective function $\mathbb{D}$ depending on the clustering criterion (*i.e.*, CS) and the cluster centers:

$$\mathbb{D} = \sum_{n=1}^{N} \sum_{s \in \mathbb{S}} y_n^s \cdot d(w_n, \mathbb{C}_s^*) \tag{6}$$

where $\mathbb{S}$ denotes all sets of clusters, and y_n^s indicates whether device n belongs to the cluster $s \in \mathbb{S}$ or not. In each epoch, the centroid of cluster s, *i.e.*, $\mathbb{C}_s^*$, is represented as the aggregated model among the devices in the same cluster. The clustering criterion between each local model w_n and the cluster center $\mathbb{C}_s^*$ is defined as $d(w_n, \mathbb{C}_s^*) = \langle w_n, \hat{\mathbb{C}}_s^* \rangle / \{||w_n|| \cdot ||\hat{\mathbb{C}}_s^*||\}$, where $\hat{\mathbb{C}}_s^*$ is the sub-model of $\mathbb{C}_s^*$ and shares the same model architecture as w_n. The detailed clustering procedure for minimizing Eq. (6) is as follows,

- *Initialization.* We select $|\mathbb{S}|$ arbitrary devices from the set of devices that train width 1×, and set their local models as the centroid $\mathbb{C}_s^*$ of each cluster s (Lines 1-2 in Alg. 1).
- *Cluster Construction.* We then calculate the CS between the local model w_n of each device n and centroid $\mathbb{C}_s^*$ of each cluster s, *i.e.*, $d(w_n, \mathbb{C}_s^*)$. Each device n will be assigned to the cluster s with $\sum_{n=1}^{N} y_n^s < M$ that minimizes $d(w_n, \mathbb{C}_s^*)$. Otherwise, when the number of devices in all clusters is larger than M, device n will be directly assigned to the cluster s with minimum $d(w_n, \mathbb{C}_s^*)$ (Lines 4-11 in Alg. 1).
- *Centroid Adjustment.* After assigning all devices to the corresponding clusters, we demand to update the centroid of each cluster. For cluster s, we update its cen-

troid as the aggregated model (Section 4.3) among the devices in current cluster (Lines 12-14 in Alg. 1).

We iteratively perform two phases of *Cluster Construction* and *Cluster Adjustment* until the cluster structure does not change or the objective function in Eq. (6) does not decrease anymore (Line 15 in Alg. 1). Generally, the number of devices is much larger than the number of widths in SNN (*i.e.*, $N \gg M$). To ensure each width will be assigned to at least one device for training, each cluster will be assigned with more than M devices in *Cluster Construction*, which is achieved by providing clusters with fewer devices a higher priority.

Width Assignment. Based on the clustering results, we then perform width assignment considering the heterogeneous training capacities and non-iid data of devices to counterbalance their performance gap and accelerate model training.

Firstly, we fine-tune the cluster structure by sorting the devices that belong to the same cluster based on their training capacities. In each epoch, the parameter server only profiles the computing and communication time of devices for the training strategy formulation of next epoch. Since devices may train models with different widths, their corresponding time consumption can not be directly used for the training capacity estimation without standardization. According to the inherent properties of each width (parameter size or computing overhead), we can easily measure the time consumption for training the width $1.0\times$ by each device. For example, given time cost t for transmitting ResNet $0.25\times$ (*i.e.*, 2.67MB), the standardized time cost for transmitting ResNet $1.0\times$ (*i.e.*, 42.61MB) can be estimated as $42.61t/2.67 \approx 16t$. Based on this, we sort all devices in the same cluster in a scheduling list by non-increasing order based on their standardized per-epoch time consumption.

Secondly, starting from the smallest width, we perform width assignment sequentially. Specifically, we travel all clusters and allocate the first device in the list of each cluster with the current minimum width for training. We then remove those devices and the assigned width from the corresponding list. On the one hand, such operation guarantees that all widths can be trained by a set of devices with similar data distributions. On the other hand, each device is assigned a width corresponding to its training capability to deal with system heterogeneity.

Finally, note that there are still $N - M|\mathbb{S}|$ (smaller than M) devices left that have not been assigned with width configuration. Since the width assignment starts from the devices with low training capacity, we can allocate the remaining high-capacity devices with any available widths without destroying training efficiency. Moreover, in order to allow high-capacity devices to fully train models with different widths, we will pick one device from each nonempty cluster and assign it a randomly selected width. We repeat such operation until all clusters become empty. At this moment, the parameter server finishes model distribution.

4.2 Local Training Optimization

On receiving the distributed model from the parameter server, each device trains model for minimizing the local loss, *i.e.*, the device iteratively feeds its local data to the model and updates the parameters using the computed gradient. To further improve the data utility and reduce the synchronization cost, FedSNN enables each device n to adapt its local training iterations τ_n (which is by default set as τ) during training independently. Specifically, as devices are usually equipped with limited and heterogeneous resources in edge computing, we leverage the training speed gap among devices, and let those faster devices perform more local iterations to see more data samples so as to increase their statistical efficiency without increasing the current epoch's duration.

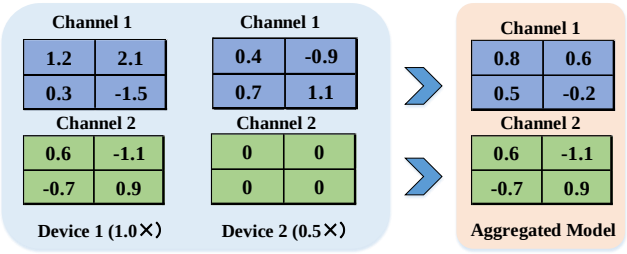


Fig. 5: Illustration of the multi-width SNN aggregation, where local iterations of two devices are identical.

For each device, its computing delay is associated with the model width, the number of local iterations and its computing capacities, while the communication delay mainly depends on the model size, and the transmission rate. Specifically, let $\phi_{n,m}$ be the per-iteration computing delay for training SNN_m and $\psi_{n,m}$ be the per-epoch communication delay for model transmission of device n. Since devices perform training in parallel, the per-epoch duration T depends on the slowest participating device, which can be calculated by $T = \max_n\{\sum_{m=1}^{M} x_{n,m} \cdot (\phi_{n,m} \cdot \tau_n + \psi_{n,m})\}$. The duration T can be calculated by the parameter server after width assignment using default τ (*i.e.*, $\tau_n = \tau, \forall n$) in each training epoch, and then distributed to devices for leading the adjustment of their local iterations. Specifically, each device n that trains width m will adjust its local training iterations τ_n as follows,

$$\tau_n = \lfloor \frac{T - \psi_{n,m}}{\phi_{n,m}} \rfloor \tag{7}$$

Hence, the local iterations of those stragglers will not be changed and those powerful devices are able to perform more local iterations and train more data samples to increase their statistical efficiency without increasing the training delay. However, as different devices may perform different numbers of local iterations, such operation will inevitably influence the aggregation performance in conventional FL. To this end, we will propose the multi-width SNN aggregation in the next section to handle this situation.

4.3 Multi-width SNN Aggregation

After each device finishes the model training, the local updates will be forwarded to the parameter server for parameter synchronization. FedSNN differs from existing FL frameworks in that devices may learn models with different widths rather than an identical structure, and their local iterations will vary with training epochs. Thus, the traditional global aggregation approach, *i.e.*, model averaging as

depicted in Section 2.1, is not suitable for multi-width SNN aggregation.

For the SNN_m, when performing model distribution and local training, the masked channels are physically removed from the original SNN so as to reduce the computing and communication cost on devices. During model aggregation, the parameter server first performs model recovery for each received SNN_m based on its corresponding binary mask Γ_m. The recovered models have the same network structure as the entire SNN, and the parameters in the masked position are all set to 0. Inspired by the traditional SNN aggregation scheme where the back-propagated gradients of all widths are accumulated to update the model parameters, *i.e.*, $w \leftarrow w - \eta \sum_{m=1}^{M} \nabla f(w \odot \Gamma_m)$, FedSNN performs multi-width SNN aggregation recursively from the smallest width. For the smallest width SNN_1, its parameters will be trained by all devices, and thus the aggregation step is formulated as $w \odot \Gamma_1 \leftarrow \sum_{n=1}^{N} a_{n,1}(w_n \odot \Gamma_1)$, where $a_{n,1}$ is the weight of model updates from device n, and $\sum_{n=1}^{N} a_{n,1} = 1$. After this, all parameters in the smallest width of the global SNN w are updated. Then, the parameters in the non-overlap part on SNN_1 and SNN_2 are updated by $w \odot (\Gamma_2 - \Gamma_1) \leftarrow \sum_{n=1}^{N} a_{n,2}(w_n \odot (\Gamma_2 - \Gamma_1))$. Accordingly, we travel all widths and update the model parameters sequentially, *i.e.*, $w \odot (\Gamma_m - \Gamma_{m-1}) \leftarrow \sum_{n=1}^{N} a_{n,m}(w_n \odot (\Gamma_m - \Gamma_{m-1})), \forall m = 2, ..., M$. For ease of understanding, we give a simple example in Figure 5.

In traditional distributed machine learning systems [43], [44], the devices are always equipped with sufficient resource and will perform model training over all its local data rather than execute several iterations using mini-batch SGD in each epoch. As more data means more information, the weight of model update from each device is proportional to the size of its local dataset during model aggregation in these systems. Inspired by this, we set the weight $a_{n,m}$ for each device n when updating width m as follows,

$$a_{n,m} = \frac{\sum_{m'=m}^{M} x_{n,m'} \cdot \tau_n}{\sum_{n'=1}^{N} \sum_{m'=m}^{M} x_{n',m'} \cdot \tau_{n'}} \tag{8}$$

The local iteration τ_n determines the size of data trained by device n in FedSNN, and the devices that train more data will be assigned with larger weights during multi-width SNN aggregation to improve their statistical efficiency.

5 Theoretical Analysis

In this section, we theoretically analyze the convergence performance of FedSNN based on its training workflow. Specifically, we first introduce some notations and assumptions. Then, we formally provide the convergence analysis.

Assume the training procedure consists of K epochs in total. We divide an SNN into M disjoint regions, and let w_m and ∇f_m be the parameters and gradients of model at region m, where $w_m = w \odot (\Gamma_m - \Gamma_{m-1})$ and all elements in Γ_0 are 0. We also adopt $\mathcal{N}_m$ to represent the set of devices that train local models contain region m, and Γ_n^k be the mask of device n at epoch k. We use α be the low bound of the number of devices in $\mathcal{N}_m, \forall m$, *i.e.*, $\alpha \geq |\mathcal{N}_m|, \forall m$, which denotes each width will be trained by at least α devices.

Assumption 1. We first make some generally used assumptions [30], [45]–[47] for FL convergence analysis as follows:

1) *Lipschitzian gradient.* The loss function f_n is with L-Lipschitzian gradients, *i.e.*,

$$\|\nabla f_n(w_1) - \nabla f_n(w_2)\| \leq L\|w_1 - w_2\|, \forall w_1, w_2, n, \tag{9}$$

where $\|\cdot\|$ is the vector L_2 normal.

2) *Bound gradient.* The squared norm of stochastic gradients is bounded by constant $\varsigma^2 > 0$, *i.e.*,

$$\|\nabla f_n(w)\|^2 \leq \varsigma^2, \forall w, n. \tag{10}$$

3) *Model error among different widths.* The parameter variance among different widths are bound by $\delta^2 \in [0, 1)$, *i.e.*,

$$\|w - w \odot \Gamma_m\|^2 \leq \delta^2 \|w\|^2, \forall w, m. \tag{11}$$

4) *Unbiased estimation on stochastic gradient.* Expectation on stochastic gradients among all devices is equal to that of global model, *i.e.*,

$$\mathbb{E}[\sum\nolimits_{n \in \mathcal{N}_m} a_{n,m} \nabla f_{n,m}(w)] = \nabla f_m(w), \forall w, m, \tag{12}$$

where $\nabla f_{n,m}$ is the gradient of device n at region m.

Upon the upper bounds in APPENDIX **??**, we finally obtain the following convergence bound after K epochs given the initialized model w^0 and $\lambda = \eta N^2 L^2 \hat{\tau}^4 / \alpha^2 \bar{\tau}^2$ for simplicity.

Theorem 1. We choose learning rate $\eta < \bar{\tau}/4\hat{\tau}^2$, and use the fact that $f(w^K) \geq 0$. Hence, the widest width obtains a convergence bound after K training epochs as follows,

$$\frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E}||\nabla f(w^k)||^2 \leq \frac{f(w^0)}{KQ_0} + \frac{Q_1}{Q_0} + \frac{Q_2 \delta^2}{KQ_0} \sum_{k=0}^{K-1} \mathbb{E}||w^k||^2$$

where we introduce three constants $Q_0 = 2\eta^2\hat{\tau}^2 - \bar{\tau}\eta/2$, $Q_1 = \lambda\eta^2\hat{\tau}(1/2\bar{\tau} + \eta L\varsigma^2)$ and $Q_2 = \lambda(1/\bar{\tau} + 2\eta L)$.

By replacing the global model w^k with $w^k \odot \Gamma_m$, the convergence bound of an arbitrary width m can be obtained, accordingly. This is because we analyze the convergence bound by dividing the model into M regions, and the error induced by different widths with non-overlap regions has already been considered. Overall, we have three key observations for the convergence bound in Theorem 1.

1) *Convergence rate.* FedSNN has linear convergence rate with respect to the number of training epochs, *i.e.*, it converges as fast as the standard FL [40] with a convergence rate of $O(\frac{1}{K})$.

2) *Model error induced by multi-widths.* Training different widths among devices incurs an error term $\delta^2 \sum_{k=0}^{K-1} \mathbb{E}||w^k||^2/K$. This term is normal in many works about model pruning [30], [48], and it implies that training fewer parameters by devices will lead to larger error on the global model, *e.g.*, when each device trains the same width, this term can be eliminated. Thus, in FedSNN, to reduce the influence of this term on the convergence performance of each width, one simple but effective solution is to ensure that each width will be trained by a sufficient number of devices (*e.g.*, α). Specifically, as shown in Eq. (**??**), the larger α represents a tighter convergence bound.

3) *Estimation error caused by non-iid data.* One important assumption (*i.e.*, *Unbiased estimation on stochastic gradient*) for

the convergence analysis holds with no doubt under iid data. But for non-iid, to ensure this assumption still hold for the convergence bound of each width, we demand to guarantee that each width should be trained on the dataset with a distribution approaching the global distribution. Otherwise, estimation error will be caused due to distribution differences.

6 Performance Evaluation

6.1 Experimental Settings

In this section, we conduct extensive experiments on the deployed physical platform to evaluate the efficiency of FedSNN. All experiments are conducted using a commonly used deep learning library [49], *i.e.*, Pytorch 1.11.0 with Python 3.8.12.

Applications and Models. We evaluate FedSNN with four classical AI applications and four DNN models.

1) *Handwritten image classification* is a task to assign digits 0-9, and uppercase letters A-Z and lowercase letters a-z to the right categories. We use the Emnist dataset [50], and it includes 814,255 images in 62 classes, of which 731,668 are for training and 82,587 are for testing. We train LeNet [51] with two widths [0.5×, 1.0×] on Emnist, and the parameter size of each width is 0.55Mb and 2.02Mb, respectively.

2) *Human Activity Recognition* application is to use inertial data from smartphone accelerometers and gyroscopes so as to target the recognition of different human activities, *e.g.*, standing, sitting, or laying down. We adopt Har dataset [52] collected from 30 individuals with 7,352 data samples for training and 2,947 for testing. To train such dataset, we adopt a customized network (called CNN for simplicity) with three convolutional layers and two fully connected layers. We define four widths [0.25×, 0.5×, 0.75×, 1.0×] in CNN, and their sizes are 8.1Mb, 32.3Mb, 72.5Mb, and 128.9Mb.

3) *Object Recognition* is a widely used technique to enable a model to recognize the objects inside a graphical input. Specifically, we adopt Cifar10 dataset [53] for the evaluation, which includes 50,000 images for training and 10,000 for testing. We train ResNet with 18 layers on this dataset, and the size of its four widths [0.25×, 0.5×, 0.75×, 1.0×] is 21.4Mb, 85.3Mb, 191.81Mb and 340.9Mb, respectively. Besides, we create a subset of ImageNet [54] with large-scale images for further evaluation, called Image100, which contains 100 out of 1,000 categories, and each sample is resized with the shape of 224×224×3. For the most complex task, we adopt a famous large model VGG16 [55], which is much larger than the sizes of the above models, to classify the images in Image100. The size of four widths [0.25×, 0.5×, 0.75×, 1.0×] for VGG16 is 135.68Mb, 539.52Mb, 1.18Gb and 2.1Gb, respectively.

Setting of Data heterogeneity. To simulate the non-iid setting, we propose to create synthesized non-iid datasets with different *class distribution skews* as in [37], [38], *e.g.*, a single user can possess more data for one class or a couple of classes than others. Concretely, p (*e.g.*, 0.5 in Section 6.3) of a unique class is divided equally for every three devices and the remaining samples of each class are distributed uniformly to other devices. Accordingly, the non-iid levels of the above datasets are denoted as 0.5 in the experiment. It is worth noting that p = 0.1 is a special case, where the distribution of the training dataset is IID for 30 devices. For fair comparisons, the full test datasets are used across all devices.

System Implementation. The deployment of the system spans two parts: 1) *The parameter server.* We deploy it on an AMAX deep learning workstation with an 8-core Intel(R) Xeon(R) CPU (E5-2620v4) and 4 NVIDIA GeForce RTX 2080Ti GPUs, and the OS is Ubuntu 16.04.1 LTS. 2) *Devices.* We use 30 devices, including 10 NVIDIA Jetson TX2, 10 NVIDIA Jetson Xavier NX and 10 NVIDIA Jetson AGX. Each TX2 has one GPU and one CPU cluster (*i.e.*, a 2-core Denver2 and a 4-core ARM CortexA57). Each NX is equipped with a 6-core NVIDIA Carmel ARMv8.2 CPU and a 384-core NVIDIA Volta GPU. The AGX carries a 512-core NVIDIA Volta GPU and an 8-core NVIDIA Carmel ARMv8.2 CPU. All devices are arranged at different places in an office and communicate with the parameter server via the 5GHz WiFi.

Setting of System Heterogeneity. To enable the devices with heterogeneous computing and communication capabilities, we present the following experimental settings.

1) *For Computation.* All the Jetson TX2, NX and AGX devices can be configured to work with different modes, which specifies the number of working CPUs and the frequency of CPU/GPU for the devices to work with different computing capacities. Specifically, TX2 can work in four modes each while NX and AGX work in one of eight modes. Devices working in different modes exhibit diverse capabilities. For instance, the AGX with highest performance mode (*i.e.*, mode 0 of AGX) achieves training by 100× faster than the TX2 with lowest performance mode (*i.e.*, mode 1 of TX2). To further reflect the time-varying on-device resources, we randomly change the modes for devices every 10 epochs.

2) *For Communication.* All devices are connected to the server via a commercial WiFi router. We arrange 30 devices into five groups, each containing 6 devices. These groups are then placed at different locations, *i.e.*, 2m, 8m, 14m, 20m and 25m away from the WiFi routers. Due to random channel noise and competition among devices, the bandwidth between the PS and devices dynamically varies during the training.

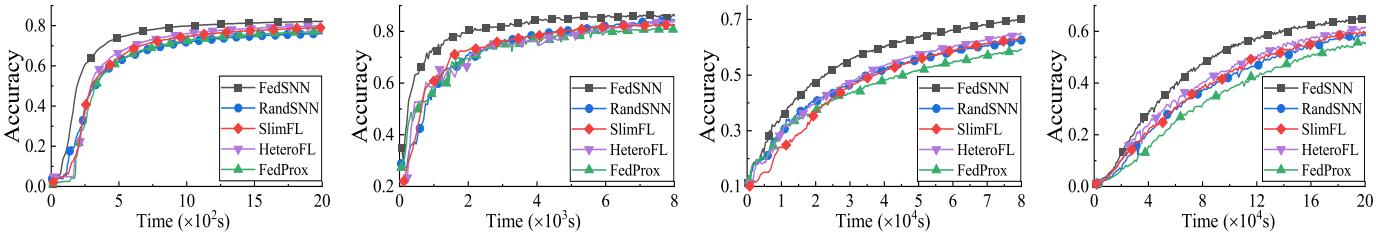
Baselines and Metrics. We evaluate the efficiency of FedSNN by comparing it with four baselines. 1) FedProx [32] is a famous FL framework viewed as a generalization and reparametrization of FedAvg [22] 2) HeteroFL [56] enables the training of heterogeneous local models with varying computation complexities. In FedProx and HeteroFL, each width of a DNN will be regarded as an independent model and be trained individually. 3) SlimFL [57] enables each device to train all widths sequentially during per-epoch local training and to forward the accumulated gradients to the parameter server for aggregation. 4) RandSNN distributes widths to devices randomly and uniformly for training in each epoch.

We compare their performance by adopting three metrics: 1) *Classification accuracy* is the proportion of correctly classified samples to all testing samples. Since each SNN has multiple widths, we take their average accuracy for ease of plotting. 2) *Completion time* is the time duration until model convergence. 3) *Network traffic* is the total size of the model transmitted through the network.

TABLE 2: Final accuracy of different frameworks on different widths and the achieved training time speedup of FedSNN.

Applications	Models	FedSNN		RandSNN		FedProx		HeteroFL		SlimFL	
		Accuracy	Speedup	Accuracy	Speedup	Accuracy	Speedup	Accuracy	Speedup	Accuracy	Speedup
Emnist	LeNet 0.5×	**80.1%**	1×	73.6%	3.36×	78.5%	**3.68**×	77.9%	2.32×	79.3%	1.99×
	LeNet 1×	**80.9%**	1×	79.2%	1.74×	80.1%	2.34×	79.5%	1.64×	78.0%	**3.79**×
Har	CNN 0.25×	85.3%	1×	83.7%	2.69×	84.3%	**4.37**×	83.9%	3.14×	**86.2%**	1.08×
	CNN 0.5×	**88.8%**	1×	85.4%	**4.59**×	87.4%	4.24×	87.2%	3.37×	87.8%	1.31×
	CNN 0.75×	**89.0%**	1×	86.2%	**5.16**×	88.3%	4.55×	88.4%	3.45×	88.2%	1.52×
	CNN 1×	**89.1%**	1×	87.1%	4.41×	88.7%	**5.11**×	88.6%	2.97×	88.8%	2.12×
Cifar10	ResNet 0.25×	**58.8%**	1×	43.2%	2.64×	56.1%	3.63×	55.1%	**3.75**×	57.2%	2.17×
	ResNet 0.5×	**63.2%**	1×	55.6%	1.92×	59.9%	2.86×	58.9%	**3.11**×	61.5%	1.37×
	ResNet 0.75×	**65.3%**	1×	59.7%	2.11×	62.3%	**3.13**×	61.8%	2.87×	63.4%	1.85×
	ResNet 1×	**66.7%**	1×	60.5%	2.31×	66.8%	**2.86**×	63.1%	2.13×	64.2%	2.54×
Image100	VGG16 0.25×	**40.7%**	1×	34.8%	2.84×	37.1%	3.23×	34.1%	**3.45**×	39.0%	1.91×
	VGG16 0.5×	**46.9%**	1×	42.1%	2.13×	42.9%	2.66×	41.7%	**3.11**×	45.7%	1.81×
	VGG16 0.75×	**49.3%**	1×	44.7%	2.41×	48.0%	2.73×	45.6%	**2.97**×	46.6%	1.98×
	VGG16 1×	**50.1%**	1×	46.4%	2.25×	49.6%	**2.96**×	47.1%	2.26×	48.3%	2.13×

* Speedup denotes the speedup of FedSNN relative to the baselines when achieving similar accuracy *w.r.t.* training time.



(a) Emnist (b) Har (c) Cifar10 (d) Image100

Fig. 6: Time vs. Accuracy for different applications with iid data.

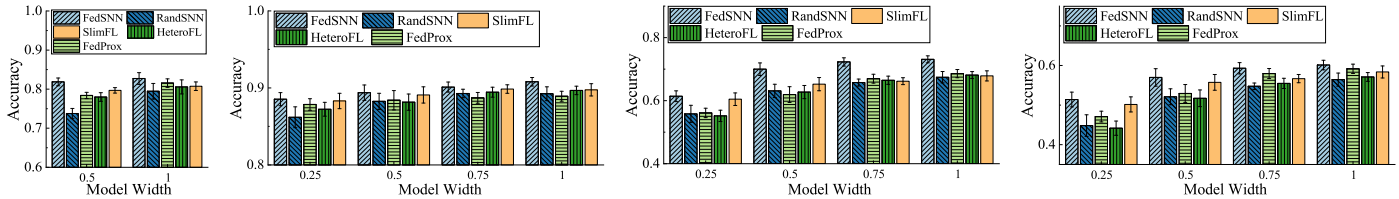
Experimental Parameters. We by default adopt the following settings in all experiments. The learning rate η for model updating is set as 0.001, 0.005, 0.01 and 0.1 for Emnist, Cifar10, Har and Image100, respectively. We also set the local iterations $\tau = 10$ for Emnist, and $\tau = 50$ for Image100, Cifar10 and Har [45], [58]. For FedSNN, we adopt $\alpha = 15$ for Emnist to ensure that each width will be trained by at least 15 devices during per-epoch training, and also take $\alpha = 7$ for the remaining three datasets.

6.2 Overall Performance

In this section, we first present the performance of all frameworks under ideal data distribution (*i.e.*, iid), where the dataset is distributed to each device randomly and uniformly. Specifically, we observe how the average accuracy changes with the training time, present the final accuracy of each width until model convergence and measure the corresponding network traffic consumption. The results are shown in Figures 6-8.

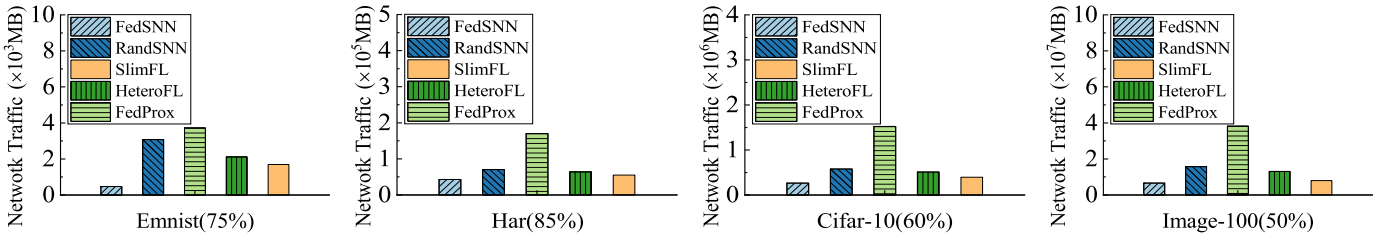
Performance of Training Acceleration. As shown in Figures 6, from the perspective of time consumption, FedSNN converges much faster than baselines, and achieves a higher accuracy given the same time budget. For Emnist, FedSNN achieves a speedup of about 3.15×, 2.35×, 2.18× and 2.91×, compared with RandSNN, SlimFL, HeteroFL and FedProx when achieving the same target accuracy. For Har, the speedup is 1.67×, 2.67×, 2.16× and 3.21×, correspondingly. For the most complex task, *i.e.*, training VGG16 on Image100, FedSNN also accelerates the training procedure by about 1.98-3.15×, which demonstrates the efficiency of our proposed framework over various training tasks. For these baselines, due to the system heterogeneity, devices with different training capacities train the same or randomly assigned widths will inevitably lead to synchronization cost, and thus slows down the training procedure. But FedSNN can make full use of devices' heterogeneous resources by performing effective model distribution, *e.g.*, we will distribute larger widths to powerful devices and enables the faster devices to train over more local data samples to alleviate the synchronization barrier caused by system heterogeneity and improve the data efficiency so as to accelerate model training.

Convergence Performance on Each Width. Figure 7 shows the detailed final accuracy on different widths of each model (*i.e.*, LeNet, CNN, ResNet and VGG16) until convergence. As the width becomes wider, the model accuracy also increases. In most cases, FedSNN outperforms baselines and obtains a higher accuracy. For example, when training Emnist, FedSNN (81.86% ± 1.03%) improves the final accuracy by 8.1%, 2.18%, 3.88%, and 3.47%, compared with RandSNN (73.76% ± 1.31%), SlimFL (79.68% ± 1.12%), HeteroFL (77.98% ± 1.19%), and FedProx (78.39% ± 0.8%) on LeNet 0.5×. For LeNet 1.0×, the improvement of accuracy is 3.25%, 2.01%, 2.34% and 1.28%, accordingly. For the remaining three datasets, in which an SNN with four widths is trained on each dataset, FedSNN also increases the model accuracy by about 1.88%-7.53%. Besides, by the results, smaller models have larger variance in accuracy while larger models have smaller variance in accuracy, indicating that the larger model is more stable. As we know, the quality



(a) Emnist (b) Har (c) Cifar10 (d) Image100

Fig. 7: Final accuracy on different widths until model convergence.



(a) Emnist (b) Har (c) Cifar10 (d) Image100

Fig. 8: Network traffic consumption for achieving target accuracy.

of local training and model aggregation determines model accuracy. Firstly, FedSNN enables faster devices to train more data so as to improve their data efficiency without delaying the training. Secondly, FedSNN also assigns these devices with higher weights during model aggregation to guarantee their statistical utility rather than averaging all local models. As a result, FedSNN achieves better convergence performance.

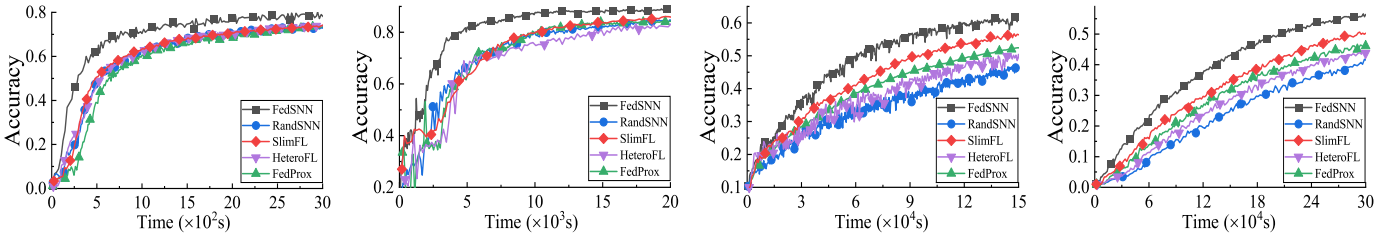
Network Traffic Consumption. In terms of the network traffic consumption, FL always consumes extremely large amount of traffic resource in edge computing, *e.g.*, 10^3-10^6MB in the experiment, but Figure 8 shows that FedSNN often consumes less network traffic resource than these baselines. For LeNet on Emnist, FedSNN (472MB) reduces the consumption of network traffic by about 84.7%, 72.2%, 77.8% and 87.1% compared with RandSNN (3,078MB), SlimFL (1,698MB), HeteroFL (2,125MB) and FedAvg (3,653MB) when achieving the same target accuracy. Furthermore, the larger the DNN size, the better the effect of FedSNN on traffic consumption reduction. For example, compared with RandSNN, FedSNN saves 2,606MB, 27,373MB, 312,524MB, and 7,125,240MB of traffic for LeNet, Har, ResNet, and VGG16, respectively. For FedProx, the achieved traffic consumption reduction is 3,352MB, 187,228MB, 959,492MB, and 21,594,920MB, correspondingly. By distributing widths to devices based on their per-epoch training results and adjusting their number of local iterations among two consecutive training epochs, FedSNN significantly reduces the total number of required training epochs for model convergence and thus greatly reduces the network traffic consumption.

6.3 Impact of Data and System Heterogeneity

To verify whether FedSNN can effectively handle data and system heterogeneity, we conduct the following experiments and the results are shown in Table 2 and Figure 9.

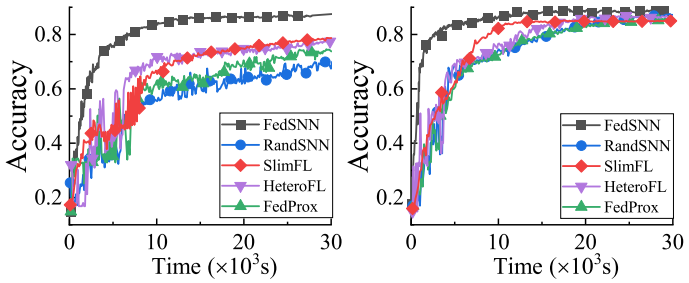
Data Heterogeneity. Table 2 presents the final accuracy of all frameworks and their corresponding speedup of training same model on data with non-iid level of 0.5, in which "Speedup" denotes the speedup of FedSNN relative to the baselines when achieving similar accuracy *w.r.t.* training time. Generally, FedSNN outperforms baselines with the improvement of final accuracy by about 0.8%-11.7%, and accelerates the training procedure by about 1.31-5.16× for four training tasks. Three observations are as follows. Firstly, although FedSNN fails to reach a higher accuracy than SlimFL on CNN 0.25×, FedSNN significantly reduces the completion time when the model converges, and achieves a higher average accuracy on four widths, *e.g.*, the average model accuracy on CNN of two frameworks is 88.05% and 87.75%, respectively. Secondly, the main difference between FedSNN and HeteroFL lies in the model distribution strategies, where FedSNN distributes widths according to the local models' features and heterogeneous training capacities among devices while HeteroFL only considers devices' model training abilities. Experimental results show that FedSNN outperforms HeteroFL, which demonstrates the efficiency of our adaptive model distribution for handling non-iid issue. For instance, compared with HeteroFL, FedSNN improves the model accuracy by 6.6%, 5.2%, 3.7%, and 3% on VGG16 0.25×, 0.5×, 0.75×, and 1×, respectively, and accelerates the training by about 3.45×, 3.11×, 2.97×, 2.26×, accordingly. Thirdly, compared with RandSNN, FedSNN optimizes the model distribution and local training procedure of each device, and consequently achieves a speedup of 1.74-5.16× when reaching the final target accuracy. These results validate the efficiency of our proposed algorithm in FedSNN compared to the random strategy, indicating that adaptive model distribution and local training optimization are helpful in alleviating data and system heterogeneity, thereby improving training performance.

System Heterogeneity. We further analyze FedSNN's robustness under higher degree of system heterogeneity. Specifically, we disable the GPUs of half of devices (*i.e.*, 5 TX2, 5 NX and 5 AGX), and the faster device can be 100× faster than the slowest one as a result. Figure 9 plots the experimental results for LeNet on Emnist, CNN on Har, ResNet on Cifar10, and VGG16 on Image100. We note that FedSNN also obtains better time-to-accuracy performance



(a) Emnist (b) Har (c) Cifar10 (d) Image100

Fig. 9: Time vs. Accuracy for different applications with data and system heterogeneity.



(a) local iteration $\tau = 10$ (b) local iteration $\tau = 30$

Fig. 10: Time vs. Accuracy for different frameworks with different numbers of local iterations on Har.

than baselines. For example, given a time budget (3,000s) for Emnist, FedSNN only spends 1,235s, 1,256s, 1,358s and 1,419s to achieve the final accuracy of RandSNN, SlimFL, HeteroFL and FedProx. For VGG16, the corresponding training speedup is 4.18×, 1.84×, 3.07×, and 2.85×. These results demonstrate that FedSNN can accelerate the training to a great extent. We further see that our framework is more robust without a significant increase in the completion time compared with baselines. For Har, FedSNN consumes 5,525s to achieve a target average accuracy (*i.e.*, 85%) under default experimental settings and the time consumption of RandSNN and SlimFL is 12,485s and 6,995s. With higher system heterogeneity, FedSNN takes only 2,142s more to achieve the same target, while RandSNN and SlimFL spend 9,574s and 12,492s more. That is because FedSNN will assign widths to devices considering their available training capacities, and let weak devices only train small widths with less resource requirement.

6.4 Hyper-Parameter Evaluation

In this section, we evaluate the sensitivity of FedSNN to the hyperparameter settings. We first investigate how the training performance changes with the default number of local iterations τ, which will influence the adjustment of devices' local training. We also examine the impact of parameter α, *i.e.*, the low bound of the number of devices that participate in each width's per-epoch training. The experimental results are shown in Figures 10, 11, and 12, and Table 3.

Impact of Number of Local Iteration τ. Figure 10 shows the training curves of all frameworks with $\tau = 10, 30$ for CNN on Har, and the results for $\tau = 50$ are already presented in the middle plot of Figure 9. We find that the value of local iteration τ can have a large effect on the

TABLE 3: The time (s) and network traffic (MB) consumption for achieving the target accuracy on Har.

	$\tau = 10$		$\tau = 30$	
	Time	Traffic	Time	Traffic
FedSNN	3373 (1.0×)	23935 (0.0%)	1267 (1.0×)	10473 (0.0%)
RandSNN	31662 (**9.4**×)	128351 (81.3%)	7550 (5.9×)	24834 (57.8%)
FedProx	24363 (7.2×)	214618 (**88.9%**)	8608 (**6.8**×)	62715 (**83.3%**)
HeteroFL	19362 (5.7×)	106241 (77.5%)	6562 (5.2×)	41657 (74.9%)
SlimFL	11998 (3.6×)	87972 (72.8%)	6238 (4.9×)	37702 (72.2%)

accuracy of all frameworks, and two observations are as follows. Firstly, we notice that FedSNN converges faster than baselines and reaches a higher accuracy regardless of the value of τ. For example, when $\tau = 10$, the accuracy of our framework is 86.9% at 30,000s, which is much higher than that of RandSNN (67.4%), SlimFL (78.7%), HeteroFL (77.5%), and FedProx (76.2%). For baselines, all devices take the same local training iterations. But for FedSNN, we will adaptively adjust the local iterations of those non-stragglers, enable them to train more data samples to increase their statistical efficiency without increasing the current epoch's duration, and assign them with larger weights during model aggregation. Consequently, FedSNN obtains better training performance. Secondly, we also observe that the performance divergence between FedSNN and baselines is shrinking as the value of τ increases. That is because a small τ always incurs a lot of communication cost due to its frequent global aggregation, resulting in a slow convergence rate on baselines without the adjustment of local iterations, and such situation can be alleviated as τ increases. However, we are not able to increase the value of τ arbitrarily. When τ gets larger, the local model of each device tends to get stuck at local optima, which inevitably leads to the decline of the global model accuracy. Besides, FedSNN still saves a large amount of resource with large τ compared with baselines. For example, Table 3 shows the time and network traffic consumption of different frameworks, and these values marked in brackets refer to the speedup or the proportion of traffic reduction achieved by FedSNN. When $\tau = 10$, FedSNN accelerates training by 3.6-9.4× and reduces the traffic consumption by 72.8-88.9%. FedSNN also achieves a speedup of 4.9-6.8×, and saves 57.8-83.3% network traffic when τ increases to 30.

Impact of Minimum Number of Participating Devices for Training Each Width. Figures 11 and 12 show the impact of parameter α on the training performance of FedSNN for LeNet on Emnist and CNN on Har, including the final accuracy on each width and their completion time. In general, for model accuracy, we note that the accuracy on each

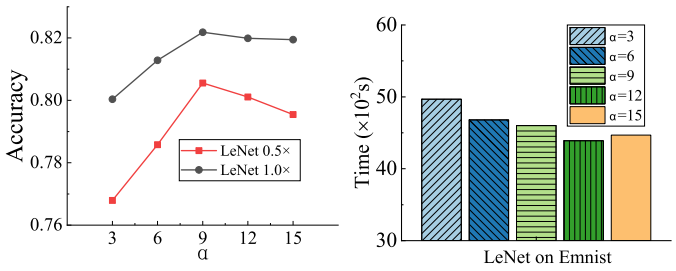


Fig. 11: Impact of parameter α for LeNet on Emnist.

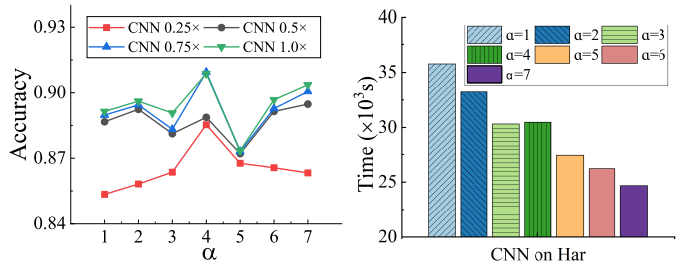
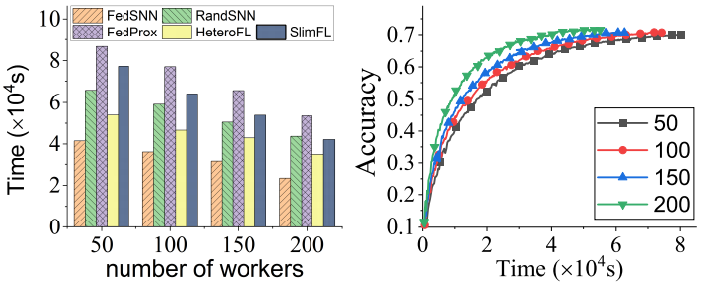


Fig. 12: Impact of parameter α for CNN on Har.

width first increases and then decreases as α gets larger. The theoretical analysis in Section 5 focuses on the convergence performance of each width, and thus proves that distributing widths to devices uniformly (*i.e.*, $\alpha \rightarrow N/M$) can ensure good convergence performance of all widths. However, due to the inherent features of SNN training, *i.e.*, training each width will inevitably interfere with the others, distributing widths to devices uniformly may not be the best solution, which offers interesting avenues for future exploration, *e.g.*, we speculate that the optimal value of α is close to $N/(2M)$ according to the experiment results. In terms of time consumption, the overall trend of completion time is that it decreases with larger α. That is because a uniform width distribution result ensures that each width can be trained sufficiently in each epoch, and thus the model converges in fewer training epochs with less time consumption. Otherwise, when we do not assign widths to devices uniformly, some widths may converge slower than the other ones, and the training procedure will not terminate until all widths converge, leading to a longer completion time.

6.5 Impact of System Scales

In this section, to demonstrate the robustness of FedSNN, we evaluate the performance of FedSNN and baselines with different scales of participating devices. We conduct several sets of experiments for ResNet on Cifar10 dataset with four scales (*e.g.*, 50, 100, 150, 200 clients) through extensive simulation experiments. We conducted the simulation experiments on an AMAX deep learning workstation equipped with an Intel(R) Xeon(R) Platinum 8358P CPU @ 2.60GHz, 8 NVIDIA GeForce RTX A6000 GPUs (48GB GPU memory each) and 512 GB RAM. The results of completion time to achieve 65% accuracy for these methods are presented in Figure 13(a), while the training processes of different scales for FedSNN are presented in Figure 13(b). As the number of participating devices increases, all methods achieve faster



(a) Completion Time (b) Training Process

Fig. 13: Performance comparison with different number of devices training ResNet on Cifar10.

convergence. The reason is that the number of data samples on a device is limited and more devices contribute more local data for model training in each round, thus speeding up model training. For instance, FedSNN with 200 devices reduce the total training time by about 31%, 25%, 11%, compared to FedSNN with 50, 100 and 150 devices, respectively. In addition, FedSNN also achieves a speedup of 1.32×∼2.25× to reach the target accuracy, compared to the baselines (*i.e.*, RandSNN, FedProx, HeteroFL, SlimFL) regarding the different system scales. These results further illustrate the robustness and advantage of FedSNN.

7 Related Works

Slimmable Neural Network. SNN was first proposed by Yu *et al.* [19] in 2018. The authors introduce a general centralized SNN training algorithm to obtain a single neural network executable at different widths. On the basis of this work, Yu *et al.* [15] further extend SNN to run at more widths using sandwich rule and inplace distillation. Previous works on SNN often take predefined width configurations for training, thus, AutoSlim [18] learns to set channel numbers in each layer of a neural network to obtain better accuracy performance. However, these methods all collect data together for performing centralized training, and force the device to train all widths sequentially with large batch size in each epoch. Nowadays, since data are often generated on end devices and widely dispersed across the network [22], distributed machine learning (*e.g.*, federated learning) can effectively prevent privacy leakage, reduce the communication cost and fully utilize the resources at the network edge [59].

Federated Learning. FL has emerged as a distributed AI approach that enables devices collaboratively to solve a learning problem [24], [43], [60]. Many previous works [59], [61], [62] show that the performance degradation always gets larger as the data and system heterogeneity increases. To enhance FL, several mechanisms have been proposed recently. For example, Wang *et al.* [45] propose a solution that adjusts the frequency of global aggregation to minimize the learning loss under a fixed resource budget. Reddi *et al.* [24] propose FedOPT, a general framework for federated optimization using server and client optimizers (including ADAGRAD, ADAM, and YOGI) to generalize many existing federated optimization methods, including FedAvg. Li *et al.* [32] propose FedProx, which can be viewed as a

generalization and re-parametrization of FedAvg to tackle heterogeneity issues. Besides, Diao *et al.* [56] propose HeteroFL to enable the training of heterogeneous local models with varying computation complexities, Then, Horvath *et al.* [63] propose FjORD, which focuses solely on adapting the model width to the client's capabilities while Alam *et al.* [64] propose FedRolex, a partial training (PT)-based approach that enables clients to train a larger global server model beyond their capabilities. The authors in [65] measure the difference between the global and local models using an experience-driven control framework, and then exclude the outlier updates to improve the training efficiency. Although there are approaches of layer-wise adaptation in FL to select certain layers for aggregation [66], [67], which is different from training SNN models with multiple widths on local devices. Furthermore, these FL schemes can only produce one global model with fixed architecture. To train SNN, they must train each width separately. But the elements of these trained widths are not shared, and devices have to deploy each width independently at the expense of more storage and management costs [16].

Incorporating FL into SNN. The related studies that incorporate FL into SNN are very limited, and the most relevant works were proposed by Baek *et al.* [20], [57], [68], [69]. SlimFL [57] mainly adjusts the local training phase in traditional FL by applying current SNN training procedure to each device. However, his studies just migrated traditional SNN training procedure to each device in FL, and the improvement (*i.e.*, superposition coding and training) is limited. For example, in superposition training, the forward propagation process of model training is basically the same as that of traditional SNN, but it will hold all widths' forward propagation losses, and then concurrently update the parameters with superpositioned gradients. The resource consumption (*e.g.*, computing and memory) on devices is tremendous as they train multiple widths simultaneously and store the corresponding gradients for parameter updating. This inevitably causes system performance degradation and brings huge synchronization barrier. On the contrary, FedSNN optimizes the main training phases in FL, so as to adapt to the characteristics of SNN and the data heterogeneity as well as system heterogeneity in edge computing and improve the training performance.

8 Conclusion

In this paper, we have proposed a general federated learning framework, namely FedSNN, to train the width-adjustable slimmable neural network by addressing data and system heterogeneity in edge computing. FedSNN redesigns the main training phases in typical FL, *i.e.*, model distribution, local training and model aggregation, to ensure all widths in SNN can be effectively trained. By applying FedSNN, the parameter server can profile the data distribution on each device based on its uploaded model features, and then distribute heterogeneous devices with different width configurations for adaptive local training. Besides, all devices' trained models will be weighted-averaged to improve their statistical utility. The experimental results show that FedSNN significantly outperforms the existing FL or SNN training schemes.

9 Acknowledgement

This article is supported in part by the National Science Foundation of China (NSFC) under Grants 62102391, 61936015, 62132019, and 62472401; in part by USTC Research Funds of the Double First-Class Initiative (Grant No. WK2150110030).

References

[1] T. Condie, P. Mineiro, N. Polyzotis, and M. Weimer, "Machine learning on big data," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 2013, pp. 1242–1244.

[2] X. Liu, Z. Yan, Y. Zhou, D. Wu, X. Chen, and J. H. Wang, "Optimizing parameter mixing under constrained communications in parallel federated learning," *IEEE/ACM Transactions on Networking*, 2023.

[3] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3642–3649.

[4] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[5] Y. Liao, Y. Xu, H. Xu, Z. Yao, L. Wang, and C. Qiao, "Accelerating federated learning with data and model parallelism in edge computing," *IEEE/ACM Transactions on Networking*, 2023.

[6] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer, "Firecaffe: near-linear acceleration of deep neural network training on compute clusters," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2592–2600.

[7] B. Ganguly, S. Hosseinalipour, K. T. Kim, C. G. Brinton, V. Aggarwal, D. J. Love, and M. Chiang, "Multi-edge server-assisted dynamic federated learning with an optimized floating aggregation point," *IEEE/ACM Transactions on Networking*, 2023.

[8] Y. Liao, Y. Xu, H. Xu, L. Wang, Z. Yao, and C. Qiao, "Mergesfl: Split federated learning with feature merging and batch size regulation," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 2054–2067.

[9] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[10] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance," *International journal of Remote sensing*, vol. 28, no. 5, pp. 823–870, 2007.

[11] M. Yuan, L. Zhang, F. He, X. Tong, and X.-Y. Li, "Infi: End-to-end learnable input filter for resource-efficient mobile-centric inference," 2022.

[12] S. Jiang, Z. Lin, Y. Li, Y. Shu, and Y. Liu, "Flexible high-resolution object detection on edge devices with tunable latency," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 559–572.

[13] M. Wang, S. Ding, T. Cao, Y. Liu, and F. Xu, "Asymo: scalable and efficient deep-learning inference on asymmetric mobile cpus," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 215–228.

[14] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1997–2006.

[15] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1803–1811.

[16] C. Li, G. Wang, B. Wang, X. Liang, Z. Li, and X. Chang, "Dynamic slimmable network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 8607–8617.

[17] T.-W. Chin, A. S. Morcos, and D. Marculescu, "Joslim: Joint widths and weights optimization for slimmable neural networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2021, pp. 119–134.

[18] J. Yu and T. Huang, "Autoslim: Towards one-shot architecture search for channel numbers," *arXiv preprint arXiv:1903.11728*, 2019.

[19] J. Yu, L. Yang, N. Xu, J. Yang, and T. S. Huang, "Slimmable neural networks," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[20] W. J. Yun, Y. Kwak, H. Baek, S. Jung, M. Ji, M. Bennis, J. Park, and J. Kim, "Slimfl: Federated learning with superposition coding over slimmable neural networks," *arXiv preprint arXiv:2203.14094*, 2022.

[21] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1–207, 2019.

[22] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, B. McMahan *et al.*, "Towards federated learning at scale: System design," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 374–388, 2019.

[23] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[24] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.

[25] D. H. Le and B.-S. Hua, "Network pruning that matters: A case study on retraining variants," *arXiv preprint arXiv:2105.03193*, 2021.

[26] M. A. Carreira-Perpinán and Y. Idelbayev, ""learning-compression" algorithms for neural net pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8532–8541.

[27] Z. Wang, H. Xu, J. Liu, H. Huang, C. Qiao, and Y. Zhao, "Resource-efficient federated learning with hierarchical aggregation in edge computing," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[28] Y. Xu, Y. Liao, H. Xu, Z. Ma, L. Wang, and J. Liu, "Adaptive control of local updating and model compression for efficient federated learning," *IEEE Transactions on Mobile Computing*, vol. 22, no. 10, pp. 5675–5689, 2022.

[29] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[30] A. Li, J. Sun, P. Li, Y. Pu, H. Li, and Y. Chen, "Hermes: an efficient federated learning framework for heterogeneous mobile clients," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 420–437.

[31] Y. Liao, Y. Xu, H. Xu, L. Wang, C. Qian, and C. Qiao, "Decentralized federated learning with adaptive configuration for heterogeneous participants," *IEEE Transactions on Mobile Computing*, 2023.

[32] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.

[33] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[34] S. Eliad, I. Hakimi, A. De Jagger, M. Silberstein, and A. Schuster, "Fine-tuning giant neural networks on commodity hardware with automatic pipeline model parallelism," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 381–396.

[35] G. Lim, J. Ahn, W. Xiao, Y. Kwon, and M. Jeon, "Zico: Efficient {GPU} memory sharing for concurrent {DNN} training," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 161–175.

[36] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[37] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non-iid data silos: An experimental study," *arXiv preprint arXiv:2102.02079*, 2021.

[38] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1698–1707.

[39] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *Advances in neural information processing systems*, vol. 33, pp. 7611–7623, 2020.

[40] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[41] M. Duan, D. Liu, X. Ji, R. Liu, L. Liang, X. Chen, and Y. Tan, "Fedgroup: Ternary cosine similarity-based clustered federated learning framework toward high accuracy in heterogeneous data," *arXiv preprint arXiv:2010.06870*, 2020.

[42] F. Sattler, K.-R. Müller, T. Wiegand, and W. Samek, "On the byzantine robustness of clustered federated learning," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 8861–8865.

[43] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[44] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching lan speeds." in *NSDI*, 2017, pp. 629–647.

[45] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 63–71.

[46] Y. Jiang, S. Wang, V. Valls, B. J. Ko, W.-H. Lee, K. K. Leung, and L. Tassiulas, "Model pruning enables efficient federated learning on edge devices," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[47] C. T. Dinh, N. H. Tran, M. N. Nguyen, C. S. Hong, W. Bao, A. Y. Zomaya, and V. Gramoli, "Federated learning over wireless networks: Convergence analysis and resource allocation," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 398–409, 2020.

[48] Z. Jiang, Y. Xu, H. Xu, Z. Wang, C. Qiao, and Y. Zhao, "Fedmp: Federated learning through adaptive model pruning in heterogeneous edge computing," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 767–779.

[49] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[50] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "Emnist: Extending mnist to handwritten letters," in *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 2921–2926.

[51] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[52] D. Anguita, A. Ghio, L. Oneto, X. Parra, J. L. Reyes-Ortiz *et al.*, "A public domain dataset for human activity recognition using smartphones." in *Esann*, vol. 3, 2013, p. 3.

[53] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[54] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[55] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[56] E. Diao, J. Ding, and V. Tarokh, "Heterofl: Computation and communication efficient federated learning for heterogeneous clients," *arXiv preprint arXiv:2010.01264*, 2020.

[57] W. J. Yun, Y. Kwak, H. Baek, S. Jung, M. Ji, M. Bennis, J. Park, and J. Kim, "Slimfl: Federated learning with superposition coding over slimmable neural networks," *IEEE/ACM Transactions on Networking*, 2023.

[58] C. Li, X. Zeng, M. Zhang, and Z. Cao, "Pyramidfl: a fine-grained client selection framework for efficient federated learning," in *ACM MobiCom '22: The 28th Annual International Conference on Mobile Computing and Networking, Sydney, NSW, Australia, October 17 - 21, 2022*. ACM, 2022, pp. 158–171. [Online]. Available: https://doi.org/10.1145/3495243.3517017

[59] M. Chen, N. Shlezinger, H. V. Poor, Y. C. Eldar, and S. Cui, "Communication-efficient federated learning," *Proceedings of the National Academy of Sciences*, vol. 118, no. 17, 2021.

[60] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated learning for internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1622–1658, 2021.

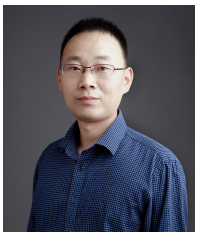
[61] Z. Ma, Y. Xu, H. Xu, Z. Meng, L. Huang, and Y. Xue, "Adaptive batch size for federated learning in resource-constrained edge computing," *IEEE Transactions on Mobile Computing*, 2021.

[62] Y. Liao, Y. Xu, H. Xu, L. Wang, and C. Qian, "Adaptive configuration for heterogeneous participants in decentralized federated learning," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[63] S. Horvath, S. Laskaridis, M. Almeida, I. Leontiadis, S. Venieris, and N. Lane, "Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout," *Advances in Neural Information Processing Systems*, vol. 34, pp. 12 876–12 889, 2021.
[64] S. Alam, L. Liu, M. Yan, and M. Zhang, "Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction," *Advances in neural information processing systems*, vol. 35, pp. 29 677–29 690, 2022.
[65] W. Luping, W. Wei, and L. Bo, "Cmfl: Mitigating communication overhead for federated learning," in *2019 IEEE 39th international conference on distributed computing systems (ICDCS)*. IEEE, 2019, pp. 954–964.
[66] J. Liu, J. Liu, H. Xu, Y. Liao, Z. Wang, and Q. Ma, "Yoga: Adaptive layer-wise model aggregation for decentralized federated learning," *IEEE/ACM Transactions on Networking*, 2023.
[67] X. Ma, J. Zhang, S. Guo, and W. Xu, "Layer-wised model aggregation for personalized federated learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 092–10 101.
[68] H. Baek, W. J. Yun, S. Jung, J. Park, M. Ji, J. Kim, and M. Bennis, "Communication and energy efficient slimmable federated learning via superposition coding and successive decoding," *arXiv preprint arXiv:2112.03267*, 2021.
[69] H. Baek, W. J. Yun, Y. Kwak, S. Jung, M. Ji, M. Bennis, J. Park, and J. Kim, "Joint superposition coding and training for federated learning over multi-width neural networks," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1729–1738.

Yang Xu (Member, IEEE) received the B.S. degree from the Wuhan University of Technology in 2014 and the Ph.D. degree in computer science and technology from the University of Science and Technology of China in 2019. He is currently an Associate Researcher with the School of Computer Science and Technology, University of Science and Technology of China. His primary research interests include the Internet of Things, edge computing, federated learning, and edge intelligence.

Yunming Liao received B.S. degree in 2020 from the University of Science and Technology of China. He is currently pursuing his Ph.D. degree in the School of Computer Science and Technology, University of Science and Technology of China. His research interests include mobile edge computing, federated learning, and edge intelligence.



Hongli Xu (Member, IEEE) received the B.S. degree in computer science from the University of Science and Technology of China, China, in 2002, and the Ph. D degree in computer software and theory from the University of Science and Technology of China, China, in 2007. He is a professor with the School of Computer Science and Technology, University of Science and Technology of China, China. He was awarded the Outstanding Youth Science Foundation of NSFC, in 2018. He has won the best paper award or the best paper candidate in several famous conferences. He has published more than 100 papers in famous journals and conferences, including the IEEE/ACM Transactions on Networking, IEEE Transactions on Mobile Computing, IEEE Transactions on Parallel and Distributed Systems, Infocom and ICNP, etc. He has also held more than 30 patents. His main research interest is software defined networks, edge computing and Internet of Thing.

Zhiyuan Wang received the B.S. degree from the Jilin University in 2019. He is currently a Ph.D. candidate in the School of Computer Science, University of Science and Technology of China (USTC). His main research interests are edge computing, federated learning and distributed machine learning.

Lun Wang received the B.S. degree in 2019 from the University of Electronic Science and Technology of China. He is currently pursuing his Ph.D. degree in the School of Computer Science and Technology, University of Science and Technology of China. His research interests include mobile edge computing and federated learning.

Jianchun Liu received the Ph.D. degree in School of Data Science from the University of Science and Technology of China in 2022. He is currently an associate researcher in the School of Computer Science and Technology at University of Science and Technology of China. His main research interests are computer networks, edge computing and federated learning.

Chen Qian received the B.S. degree from Nanjing University in 2006, the M.Phil. degree from The Hong Kong University of Science and Technology in 2008, and the Ph.D. degree from The University of Texas at Austin in 2013, all in computer science. He is currently an Assistant Professor with the Department of Computer Engineering, University of California at Santa Cruz. His research interests include computer networking, network security, and Internet of Things. He has authored over 60 research papers in highly competitive conferences and journals. He is a member of the ACM.