IoTSL: Toward Efficient Distributed Learning for Resource-Constrained Internet of Things

Xingyu Feng, Chengwen Luo, *Member, IEEE*, Jiongzhang Chen, *Graduate Student Member, IEEE*, Yijing Huang, *Graduate Student Member, IEEE*, Jin Zhang, Weitao Xu, *Member, IEEE*, Jianqiang Li, and Victor C. M. Leung, *Life Fellow, IEEE*

***Abstract*—Recently proposed split learning (SL) is a promising distributed machine learning paradigm that enables machine learning without accessing the raw data of the clients. SL can be viewed as one specific type of serial federation learning. However, deploying SL on resource-constrained Internet of Things (IoT) devices still has some limitations, including high communication costs and catastrophic forgetting problems caused by imbalanced data distribution of devices. In this article, we design and implement *IoTSL*, which is an efficient distributed learning framework for efficient cloud-edge collaboration in IoT systems. *IoTSL* combines generative adversarial networks (GANs) and differential privacy techniques to train local data-based generators on participating devices, and generate data with privacy protection. On the one hand, *IoTSL* pretrains the global model using the generative data, and then fine-tunes the model using the local data to lower the communication cost. On the other hand, the generated data is used to impute the missing classes of devices to alleviate the commonly seen catastrophic forgetting phenomenon. We use three common data sets to verify the proposed framework. Extensive experimental results show that compared to the conventional SL, *IoTSL* significantly reduces communication costs, and efficiently alleviates the catastrophic forgetting phenomenon.**

***Index Terms*—Generative adversarial networks (GANs), privacy protection, resource-constrained Internet of Things (IoT) devices, split learning (SL).**

I. Introduction

UBIQUITOUS intelligent devices are tightly connected by mobile networks nowadays to construct wireless networked Internet of Things (IoT) systems. Novel IoT applications have emerged in different aspects of modern life, forming novel intelligent systems, such as smart home systems [1], smart healthcare systems [2], etc. IoT devices close to users contribute different sensory inputs, which can be easily uploaded and accessed by cloud users through mobile networking. To extract useful models from sensor data and enable different artificial intelligence (AI) applications, corresponding machine learning models can be trained on the cloud over the sensor data to provide pervasive intelligent services. Using the mobile devices attached to users, users can at any moment access intelligent services through cloud computing or fog computing technologies (edge computing) [3].

However, in such a cloud-centric paradigm, IoT devices upload local private data to the central server for model training, which not only incurs substantial communication costs but also poses severe privacy threats [4]. Recently, as the computational power and storage capacity of embedded IoT devices gradually increases, an appealing alternative is both to keep the private data and to train the machine learning models locally on the device, which is known as the embedded AI paradigm [5]. To improve the training and inference performance on the device, popular neural network surgery techniques, such as pruning [6] and quantization [7], have been proposed to achieve efficient light-weighted network models for hardware-constrained IoT devices. However, insufficient data samples from individual devices still make it challenging for a single device with only local data to train efficient edge-side models for comprehensive AI services [8].

To solve the above problem, one solution is to produce a global model by aggregating local models from all devices while avoiding revealing any sensitive local private information. This goal can be achieved by the distributed machine learning framework, e.g., federated learning (FL) [9]. Based on the training methods used, FL can be defined into two categories: 1) parallel FL (such as the FedAvg [10]) and 2) serial FL (such as split learning (SL) [11]). The training process of parallel FL is illustrated in Fig. 1, in which all participating clients train models locally in parallel and then obtain a global model by aggregating the parameters of all these local models. Parallel FL replaces the fusion of local data by the aggregation of local models, thus, privacy is protected while enabling model performance not to be degraded by insufficient training samples from a single device. A widely accepted assumption is that the local models must have the same network architecture as the global model in order to aggregate all local models [12]. However, IoT devices are usually hardware resource-constrained, and local models are usually light-weighted. Under the same network architecture constraint, the complexity of the global model is significantly

Manuscript received 22 September 2022; revised 27 December 2022; accepted 3 January 2023. Date of publication 10 January 2023; date of current version 23 May 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 61972263 and Grant 62073225; and in part by the Stable Support Plan for Higher Education Institutions in Shenzhen under Grant 20200810113310001. *(Corresponding author: Chengwen Luo.)*

Xingyu Feng, Chengwen Luo, Jiongzhang Chen, Yijing Huang, Jin Zhang, Jianqiang Li, and Victor C. M. Leung are with the National Engineering Laboratory for Big Data System Computing Technology, and College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: fengxingyu2017@email.szu.edu.cn; chengwen@szu.edu.cn; chenjiongzhang2020@email.szu.edu.cn; hyjjeymy@szu.edu.cn; jin.zhang@szu.edu.cn; lijq@szu.edu.cn; vleung@ieee.org).

Weitao Xu is with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: weitaoxu@cityu.edu.hk).

Digital Object Identifier 10.1109/JIOT.2023.3235765

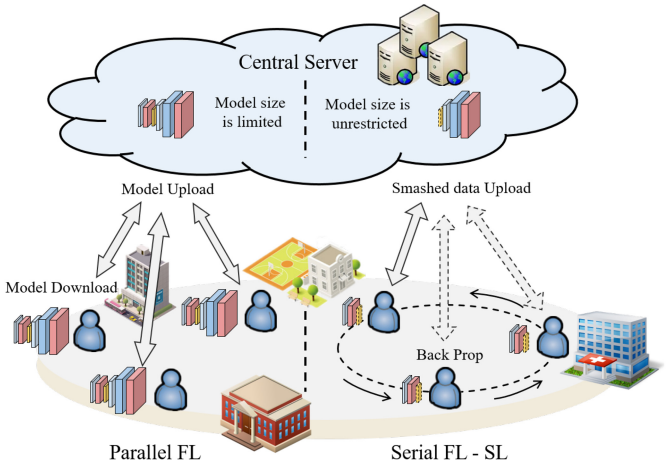


Fig. 1. Different training paradigm for parallel FL and serial FL (SL).

limited, hence, reducing the capability of the AI services provided by the cloud. Therefore, in the application scenarios of IoT, the parallel FL framework is not sufficient, it limits the global model size, adding constraints on the maximum size of the models that can be deployed on the central server.

In contrast, a serial FL framework, such as SL, addresses the above problem. The training approach of SL is illustrated in Fig. 1. The complete neural network model is split vertically into two parts, which are deployed on the device side and the central server, respectively. In the training process, each participating device collaborates with the central server and takes turns updating the global model. The local data on each device is only fed into the device's local model, and the output of the truncated layer (known as the smashed data) is sent to the central server and fed into the subsequent model for central model parameter update. Similarly, the gradient information is again sent back from the server to the device and is used to update the parameters of the device model. In this way, the SL framework splits the model structure, and the device is deployed only a small portion of the model structure instead of the entire model. The computational burden of the device is significantly reduced while uploading (or transferring) the smashed data also effectively avoids privacy leakage. Since the size of the network model on the central server is no longer limited, a larger model is now possible on the central server to provide more sophisticated AI services. Therefore, with a small device-side model to fit into the hardware-constrained IoT device, a large model on the central server to provide intelligent service, and together with privacy-preserving features, SL becomes a promising paradigm for distributed machine learning on IoT devices.

However, there are still some significant challenges remain that hinder the direct deployment of SL on IoT devices for efficient distributed machine learning.

1) *Extensive Communication Cost:* Since the model is divided into two parts and placed on both the device and central server, each model update in the training process requires communication between the device and the central server. This inevitably incurs high communication costs and energy consumption.
2) *Potential Catastrophic Forgetting:* In common application scenarios, due to factors, such as geographic location differences or user preference differences, local data on different IoT devices are typically nonindependent identically distributed (non-iid) [13]. A common problem due to non-iid is that the classes of the training data of each device can be different, resulting in the degradation of the classification performance. In serial framework SL, multiple devices take turns communicating with the central server in the model update. When the local data classes of devices differ, the updated model continues to learn new knowledge, while forgetting previously learned knowledge due to unbalanced class distributions on different devices, which is known as catastrophic forgetting [14].

To address the above challenges, in this article, we propose *IoTSL*, an efficient distributed learning framework for IoT. *IoTSL* is designed to retain all the advantages of SL while addressing the problems of SL on cloud-edge collaboration for IoT. Specifically, IoT devices are deployed with only a small part of the model, thus, reducing the computational cost on the device side. Furthermore, *IoTSL* reduces the communication cost between IoT devices and the central server in the SL training process, and also alleviates the catastrophic forgetting problem, especially, when the data on IoT devices is non-iid. To achieve this, we combine the generative adversarial networks (GANs) to train data generators for each participating device. First, we use generative data to pretrain the model in order to reduce communication costs. Second, a generator-sharing strategy is designed to effectively improve the classification performance even with non-iid data, which successfully alleviates the catastrophic forgetting phenomenon in the training process. Finally, to ensure privacy protection, a privacy-preserving module is seamlessly integrated with the system to protect the privacy of the local data of each device. The contributions of this article are summarized as follows.

1) We design and propose *IoTSL*, a distributed machine learning framework that enables efficient collaboration between resource-constrained IoT devices and the central server that retains all advantages of SL while significantly reducing communication costs and energy consumption.
2) *IoTSL* proposes a GAN-based approach to successfully address the non-iid problem in cloud-edge collaboration and alleviate the catastrophic forgetting problem, while also achieving privacy protection in collaboration.
3) We implement the system on embedded devices and design extensive evaluations to study the performance of *IoTSL*. Results of which show that *IoTSL* achieves improved accuracy compared with state-of-the-art, while also being power-efficient.

The remainder of this article is organized as follows. In Section II, we detail the technical background. In Section III, we summarize and analyze the problems that need to be addressed in the system design. In Section IV, we describe the framework of *IoTSL* and provide a detailed introduction to each module. Section V provides

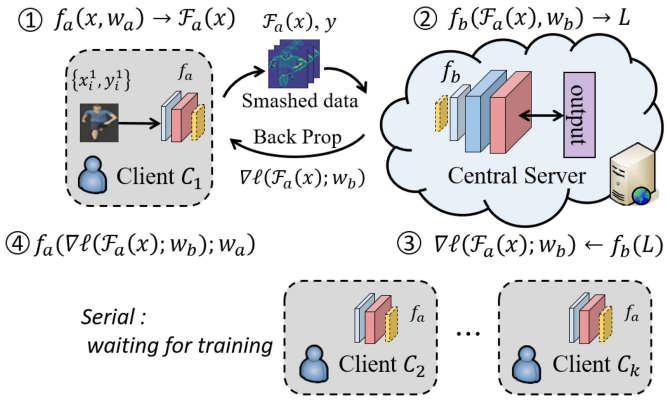


Fig. 2. Specifics of a canonical SL framework.

detailed evaluations. Section VI discusses the relative work. Section VII summarizes the limitations and future work of this article, and finally, we conclude in Section VIII.

II. Technical Background

In this section, we describe the training process of the SL framework in detail. Fig. 2 shows the canonical SL framework. We assume that there are K clients and denote them as clients $= [C_1, C_2, \ldots, C_k]$. Each client C_i has its own local labeled data $\boldsymbol{X}_i := \{(x_i^j, y_i^j)\}^{N_i}$, where x_i^j is the jth training sample on C_i, y_i^j is its corresponding ground truth label, and N_i is the total number of samples.

As can be seen from the figure, DNN model $\boldsymbol{f}$ is vertically divided into $\boldsymbol{f}_a$ and $\boldsymbol{f}_b$, which are placed on the clients and the server, respectively. Here, we assume the corresponding parameters are w_a and w_b for these two parts of the model. The network structure of the model part $\boldsymbol{f}_a$ is the same for each client. The training process of SL typically lasts for multiple rounds. In each round, all participating clients would collaborate with the server and each client would participate in training for one epoch according to the ordering. The specific process consists of the following four steps.

1) $\boldsymbol{f}_a(x, w_a) \to \mathcal{F}_a(x)$: In forward propagation, the local data x_i^j is fed into the client's $\boldsymbol{f}_a$, and the output is called *smashed data* $\mathcal{F}_a(x)$. Then, the smashed data and ground truth label y_i^j of this sample are sent to the server.
2) $\boldsymbol{f}_b(\mathcal{F}_a(x), w_b) \to L$: The smashed data continues to be fed to $\boldsymbol{F}_b$ to obtain the predicted label $\hat{y}_i^j$ and calculates the cross-entropy loss with y_i^j.
3) $\nabla \boldsymbol{l}(\mathcal{F}_a(x); w_b) \leftarrow \boldsymbol{f}_b(L)$: The model performs backpropagation to update the weights W_b of the $\boldsymbol{f}_b$ part and the central server sends the gradient back to the client.
4) $\boldsymbol{f}_a(\nabla \boldsymbol{l}(\mathcal{F}_a(x); w_b); w_a)$: The client receives the gradient information and continues to perform backpropagation on $\boldsymbol{f}_a$ to update the weights w_a.

The local data of client C_i is divided into multiple batches, and each batch would perform the above four steps to update the model until C_i has no new training data. Then, the next client C_{i+1} would continue co-training with the server. When all clients have completed training with the server, the next training round is started again until the model converges.

TABLE I
Data Transmission per Device Within One Epoch

Processes 1	Processes 2	Processes 3
$N_i * F_i + N_i$	$N_i * F_i$	w_i

III. Problem Analysis

In this section, we analyze the infeasibility of directly deploying SL on IoT devices for distributed machine learning. First, we analyze the communication cost problem of SL theoretically. Second, we evaluate the negative effects of catastrophic forgetting on classification accuracy.

On the client side, the model part $\boldsymbol{f}_a$ is typically designed as a light-weighted architecture to fit into the resource-constrained edge devices, such as Raspberry Pi, smartphones, and so on. These edge devices have basic computational power and storage capacity to support sensory data collection and data processing. The servers in the system can be smart gateways, high-performance computers or remote cloud servers, etc. Due to the unique training paradigm of SL, the following two problems exist.

Problem 1 (Extensive Communication Cost): In the training process of SL, the devices frequently transmit smashed data to the server and receive gradient updates, which results in high communication costs.

To verify the communication costs, we theoretically count the number of parameters transmitted at the device side during the training process, which consists of two dimensions. The first one is the number of parameters transferred between the device and the server in one training epoch, which is determined by the amount of local data and the size of the smashed data. The second one is the number of times the device is involved in the training, i.e., the value of rounds. In one round, all devices will train one epoch with the server, so a round contains K epoch (K is the number of participating training devices).

We use the following notation: N_i is the amount of data stored on the device $\mathcal{D}_i$, F_i is the size of smashed data on the device $\mathcal{D}_i$, and w_i is the model parameters on the device $\mathcal{D}_i$.

In Table I, we measure the amount of data transmissions for a device within one epoch. It consists of three communication steps: 1) sending smashed data to the server; 2) receiving gradients from the server; and 3) sending model parameters to the next device. Each data corresponds to a label and generates the corresponding smashed data, indicating that the transmission volume of Process 1 is $N_i * F_i + 1 * N_i$. The gradient information from the server is used to update the parameters of the device, which maintain the same size as the smashed data, so the transmission of Process 2 is $N_i * F_i$. When this device's training epoch ends, it sends the updated model parameters to the next participating device, so the transmission of Process 3 is w_i. Assume that there are K participating devices in the training task and the model requires n training rounds to reach convergence. In this training, the following parameters are communicated:

$$C_{\text{total}} = n * K * [N_i * (2F_i + 1) + w_i]. \tag{1}$$

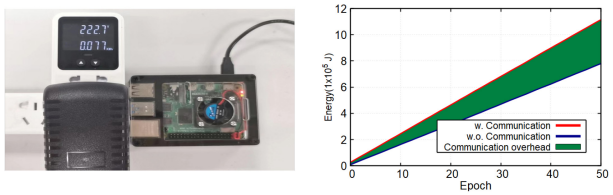


Fig. 3. Monitoring the power consumption of the Raspberry Pi using a plug-in power meter. (a) Real-time monitoring by a plug-in power meter. (b) Comparison of power consumption.

To demonstrate the energy consumption of communication, we use a Raspberry Pi 4b as an IoT device on which we deployed two convolutional layers for energy measurement, with the size of the smashed data being $64 \times 14 \times 14$. We measure the energy consumption of the device during training using a plug-in power meter [Fig. 3(a)], and the results are shown in Fig. 3(b). The red curve represents the energy consumption of the device in the training mode, and the blue curve represents the training only locally on the device (we used randomly generated gradients instead of real gradients for the measurement). It can be seen that the training process with the communication module significantly increases energy consumption, and the green part of the figure represents the energy consumption incurred, which is significantly increased by 42.6% due to communication overhead.

Problem 2 (Causes of Training Failure—Catastrophic Forgetting): The local data collected by the onboard sensors of different IoT devices can vary [15]. This variability might also cause differences in the classification classes of each device. As different devices have different classes of local data, catastrophic forgetting can occur during the serial SL training process.

Due to the setting of SL, the training process is carried out cyclically across all devices. This approach allows the model to revisit previous knowledge again when it is about to forget, which aims to alleviate catastrophic forgetting. However, this relies on the assumption that the local data distribution across devices is relatively close to ensure that the model does not forget quickly within an epoch. We conduct the following experiment to real the negative effect of catastrophic forgetting in SL with imbalanced data distribution among devices.

We use handwritten digital images from the MNIST data set [16] as an illustration. As shown in Fig. 4, in Experiment 1, we assigned 1000 c_1 images and 1000 c_2 images to device $\mathcal{D}_1$ as local data, and 1000 c_3 images and 1000 c_4 images to device $\mathcal{D}_2$. The local data distribution $D_1(x)$ for device $\mathcal{D}_1$ differs significantly from that of device $\mathcal{D}_2$. In Experiment 2, we simulate a uniform distribution of local data. We send 500 images of each type to devices $\mathcal{D}_3$ and $\mathcal{D}_4$ as local data. Therefore, data distributions of $D_3(x)$ and $D_4(x)$ are similar. To evaluate the convergence of the model in both cases, we customized a vanilla network with five convolutional layers, with the first two layers of the network placed on the device and the last three layers placed on the server. And the same testing data set is used to evaluate the accuracy, with 100 image data for each of the numbers "1" to "4," for a total of 400 images.

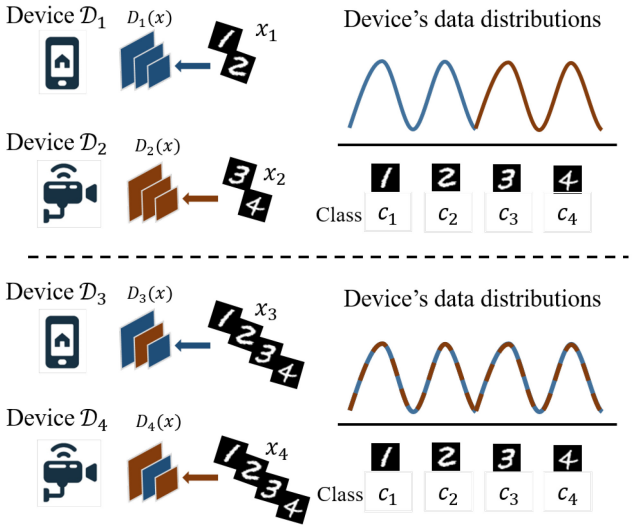


Fig. 4. Local data distribution of devices.

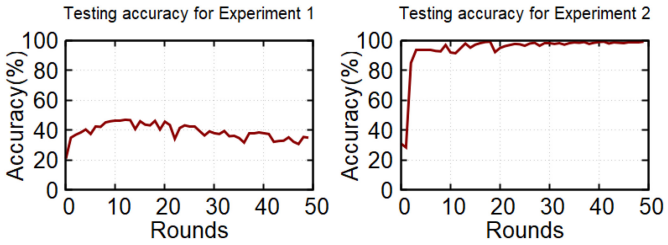


Fig. 5. Comparison of testing accuracy under different data distributions.

As shown in Fig. 5, the testing accuracy in Experiment 1 remains below 50%, indicating that the model training fails. The testing accuracy in Experiment 2 is stable at around 99%, indicating that the model can be properly trained. The experimental results show that SL is extremely sensitive to data distribution, and the non-iid phenomenon continues to cause catastrophic forgetting in SL.

IV. *IoTSL* Framework

In this section, we describe the details of the *IoTSL* framework. *IoTSL* is deployed on the central server and clients (device side), a set of n IoT devices $\boldsymbol{\mathcal{D}} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$ with local data are ordered according to collaborate with a central server $\boldsymbol{\mathcal{S}}$ to train the model.

To reduce the communication cost during training, *IoTSL* aims to reduce the number of rounds in which devices participate in training, i.e., the model is expected to converge faster. Initialized DNN models typically require multiple epochs of training to achieve convergence. Inspired by transfer learning [17], we motivate this DNN model to directly fine-tune a model that has been trained for a similar task, relying on previously learned knowledge to solve new problems [18], thus, speeding up convergence. However, the AI services available in various IoT scenarios are diverse, making it difficult to obtain a model for a similar task directly. To obtain the pre-trained model, we use GANs [19] to train a local data-based generator on each participating device, and then use the generated data to train the model. Simultaneously, the generated

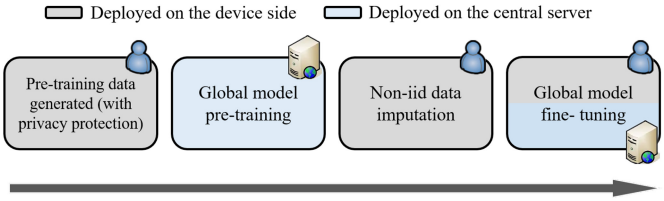


Fig. 6. Workflow of *IoTSL*. The gray module is deployed on the device side and the blue module is deployed on the central server.

data can be used to impute the non-iid local data of the devices, thereby alleviating the catastrophic forgetting phenomenon. In summary, *IoTSL* adds four modules to the traditional SL framework, as shown in Fig. 6: Pretraining data generation module, Global model pretraining module, non-iid data imputation module, and Generator privacy protection module. In the following sections, we provide the detailed design rationale for each module.

A. Pretraining Data Generation

As the first step in *IoTSL*, we train the generative models based on local data distribution on each participating device to generate data samples for each device. Compared to models, such as VAE [20], Flow-based [21], and other generative models, due to the introduction of adversarial networks, GANs are more efficient in generating high-quality data samples. In *IoTSL*, since generated data will be used for model training, the quality of the generated data is crucial, and using GANs in *IoTSL* provides sufficient performance as evaluated later in Section V.

GANs consist of a generator and a discriminator, which are two independent models. In *IoTSL*, we use the local data x_i on the device $\mathcal{D}_i$ as the original data for the above two models, where the generator takes noise as input and generates synthetic data (fake data) by capturing the distribution of the original data x_i. The discriminator takes the output of the generator (fake data) and the original data x_i as input, learns to distinguish the differences in their data distributions and outputs a score that would be used to determine whether the input data is true or fake. The generator continuously improves its ability to generate fake data, and it uses the discriminator's score as a criterion for model updates. The discriminator also keeps improving its ability to identify true and fake data until the generator can generate data that is fake to be true.

The interactive training of generators and discriminators is contained in the following objective function:

$$\begin{aligned}&\min_G \max_D V(D,G)\\&= \mathbb{E}_{x\sim p_{\text{data}}(x)}\big[\log(D(x))\big]\\&\quad + \mathbb{E}_{z\sim p_z(z)}\big[\log(1-D(G(z)))\big]\end{aligned} \tag{2}$$

where G is the generator, D is the discriminator, z is the input noise of the generator, $p_z(z)$ is the distribution of the noise, x is the local true sample, and $p_{\text{data}}(x)$ is the distribution of the true sample.

In the subsequent modules of *IoTSL*, the generated data produced by the generator would be used to impute or replace the local data on the device to perform supervised training tasks. The GANs approach described above only requires the generator to fit the distribution of the original data without considering the label information of the samples, resulting in unlabeled generated data. Due to the lack of label information, these data cannot be used directly for supervised learning training tasks or accurately impute the missing classes of other devices. As a result, we use conditional GAN [22] in *IoTSL*, which is an extension of GAN. To ensure that the generated samples can match the currently set label conditions, both the generator and the discriminator are conditional on the class label information. The training objective function for conditional GAN is as follows:

$$\begin{aligned}&\min_G \max_D V(D,G)\\&= \mathbb{E}_{x\sim p_{\text{data}}(x)}\big[\log(D(x|y))\big]\\&\quad + \mathbb{E}_{z\sim p_z(z)}\big[\log(1-D(G(z|y)))\big]\end{aligned} \tag{3}$$

where y is the label information of samples.

In *IoTSL*, the generator deployment has two advantages. First, the pretrained generators can be easily passed among IoT devices to generate data samples for each device at any time. Therefore, when IoT devices need to exchange a large amount of data with each other, sharing generators instead of the raw data can significantly reduce the device's energy consumption and communication overhead. Second, the generated data can be used as a substitute for the device's local data which alleviates the need for sharing the original private local data (detailed in Sections IV-B and IV-C), thus, avoiding direct privacy leakage.

B. Global Model Pretraining

In *IoTSL*, we deploy a generator based on local data for each participating device, corresponding to its order, and the generators are denoted as $G = \{G_1, G_2, \ldots, G_n\}$. Since the generated data has a similar distribution to the local data, it is suitable to be used as pretrained training data.

After obtaining the pretrained model, the formal training can achieve convergence only by fine-tuning the pretrained model, thereby reducing the number of training rounds and the communication cost, which was the original purpose of introducing pretraining. Obviously, there should be no additional communication costs associated with the pretraining process. However, the model is split between the devices and the server, and pretraining also requires bilateral cooperation. One assumption is that instead of the distributed method, we deploy the pretraining process on one side (device side or central server). In general, the server is considered to be resource-unrestricted. Although the pretraining data is provided by a generator on the device, the generator can be shared with the server. Instead of directly transferring large amounts of pretraining data, generators can generate data on the server. We redesigned the tasks of the device side and the server and moved the pretraining process entirely to the server.

According to the setting of SL, only the second half of the model is deployed on the server, whereas pretraining, here, should target the global model. Therefore, we redefine the

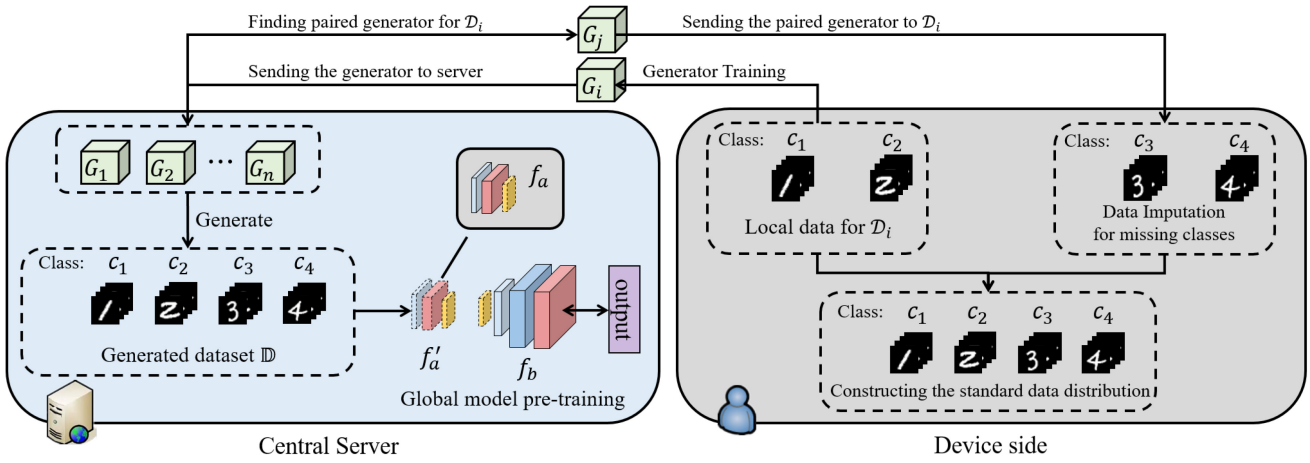


Fig. 7. Overview of the global model pretraining module and the non-iid data imputation module.

structure of the model on the server. As shown in Fig. 7, we add the device model part $\boldsymbol{f}_a$ to the server and splice it with $\boldsymbol{f}_b$ to form the complete network model $\boldsymbol{f}$. Another critical step is to allow the central server to acquire the pretraining data set $\mathbb{D}$. In *IoTSL*, each participating device sends its generator G_i and labeling information of local data to the server. According to the above principle of conditional GANs, the generator G_i can generate data that meets the labeling conditions. With reference to the local data distribution of each device $\mathcal{D}_i$, we use G_i to generate data on the same scale as the local data, i.e., the same classes and the number of samples in each class as the local data while saving the generation records of each generator to the log file. All generated data are then combined into a data set $\mathbb{D}$ by class, and this approach makes the distribution of the data set $\mathbb{D}$ more similar to the fusion of all local data. It is worth noting that the ground truth label y of the device's local data would be sent to the server as the condition to generate the data, but this does not violate the privacy-preserving definition of SL. This is because, during the training process of SL, y is allowed to be exposed to the server, and it would be used for computing the gradient.

Finally, we use the data set $\mathbb{D}$ to train the global model on the server. After the model converges, the server sends $\boldsymbol{\theta}_{\text{pre},a}$ (the parameters of $\boldsymbol{f}_a'$) to the first participating device to update $\boldsymbol{\theta}_a$ (the parameters of $\boldsymbol{f}_a$), instead of randomly initializing $\boldsymbol{\theta}_a$ on first training epoch. With the above approach, the model parts on the devices and the server are pretrained. Although the transfer of the generator and $\boldsymbol{\theta}_{\text{pre},a}$ inevitably incur additional communication costs, it is a one-time transfer that can be completed offline. It is acceptable in comparison to the reduction in communication costs. We evaluate the overall efficiency of this method in Section V.

C. Non-IID Data Imputation

Although the approach described above effectively reduces the communication costs during training, non-iid local data can still lead to catastrophic forgetting, resulting in model training failure. To mitigate the negative effects of this phenomenon, we impute the non-iid data to construct a standard data distribution on each device. Specifically, we achieve this by pairing the participating devices and using each other's generators to generate data for the missing classes. We describe the pairing approach in detail.

Assume that the sum of the local data classes for all devices is M, with the label set as $L = \{l_1, l_2, \ldots, l_m\}$. For a certain device $\mathcal{D}_i$, its label set is L_i ($L_i \subseteq L$), and its generator G_i can also generate data with the label set L_i. The data set $\mathbb{D}$ is constituted of generated data from all generators and contains data from all classes, i.e., $L_{\mathbb{D}} = L$. During the pretraining preparation phase, we save the generation records of each generator in the log, which contains the classes of data generated by each generator (the corresponding label set is L_i) and the number of data generated for each class. Thus, by comparing $L_{\mathbb{D}}$ with L_i, we can determine whether the local data of device $\mathcal{D}_i$ is missing classes. If $L_i = L_{\mathbb{D}}$, it means that the device $\mathcal{D}_i$ has data from all classes and is considered as an eligible participant for SL. If $L_i \leq L_{\mathbb{D}}$, it means that some classes are missing from the local data of device $\mathcal{D}_i$. The device $\mathcal{D}_i$ is defined as defective, which may lead to catastrophic forgetting. Furthermore, we considered the balance of the number of samples in each class. Although the device contains data from a certain class, the number of samples in that class is significantly lower than in other classes, which could lead to forgetting, and the device is also defined as defective.

For defective devices, as shown in Fig. 7, we send generators of other devices to them, prompting them to generate data locally to impute data of the missing classes. Even before that, We would locate the appropriate generator for the defective device. First, we count the labels of the missing classes for the device $\mathcal{D}_i$ by $L' = L_{\mathbb{D}} - L_i$. Then, we retrieve the generation record log on the server and search for a generator G_j that can generate data for the class L' (the condition is $L_j \supseteq L'$). The device $\mathcal{D}_j$ is a paired member of $\mathcal{D}_i$. Finally, the server sends the generator G_j to the device $\mathcal{D}_i$, and G_j generates data for the missing classes. To avoid class imbalance, the generated quantity is the average of each class in $\mathcal{D}_i$'s local data. We establish a new data set from the generated data and the local data to participate in the subsequent training. In particular, the pairing relationships are one-to-many when the local data

is extremely non-iid. To form the standard data distribution, the defective devices necessitate the use of multiple device generators.

D. Enhancing the Privacy Protection of the Generators

Privacy protection of local data is crucial, especially, in IoT scenarios. *IoTSL* proposes the generator-sharing mechanism, which avoids local data sharing with other parties. However, recent researches have shown that GAN models are vulnerable to attacks, such as model inversion attacks [23]. New techniques can reconstruct training samples based on the generated data [24]. In *IoTSL*, the training samples are private data of the devices, and to further remove the risk, we introduce a privacy protection module to *IoTSL*.

The purpose of the privacy protection module is to ensure that the update process of the GAN model respects the differential privacy constraint [25], i.e., that the training samples cannot be reconstructed from the model parameters and generated data. For this purpose, we use the Gaussian mechanism to manipulate the gradients in backpropagation [26]. First, we reduce the degree of influence that each training sample x_i has on the model update. Because the weight update is dominated by the gradient of the loss, we limit the gradient $\boldsymbol{g}$ to a finite range $[-C, C]$: $\max(||\boldsymbol{g}||_2) \leq C$ (L2-norm used in the Gaussian mechanism), i.e., the original gradient generated by x_i is clipped: $\boldsymbol{g} = \boldsymbol{g}/\max(1, [||\boldsymbol{g}||_2/C])$, C is defined as the clipping bound. Then, to introduce randomness and obscure the differences in model updates from various training samples, we add the Gaussian noise $\mathcal{N}(0, \sigma^2 C^2 \boldsymbol{I})$ that is positively correlated with C to the clipped $\boldsymbol{g}$. In summary, the gradient sanitization mechanism $\mathcal{M}_{\sigma,C}(\boldsymbol{g})$ is defined as

$$\mathcal{M}_{\sigma,C}(\boldsymbol{g}) = \text{clip}(\boldsymbol{g}, C) + \mathcal{N}\left(0, \sigma^2 C^2 \boldsymbol{I}\right) \tag{4}$$

where σ is the noise multiplier.

In *IoTSL*, since the discriminator is not sent to other sides, we only sanitize the generator's gradient, while the discriminator retains the original gradient update method. The update process for both is as follows:

$$\begin{cases} \boldsymbol{\theta}_D^{t+1} = \boldsymbol{\theta}_D^t - \eta_D \cdot \boldsymbol{g}_D^t \\ \boldsymbol{\theta}_G^{t+1} = \boldsymbol{\theta}_G^t - \eta_G \cdot \mathcal{M}_{\sigma,C}(\boldsymbol{g}_G^t) \end{cases} \tag{5}$$

where θ_G and θ_D are the model parameters of the generator and discriminator, and η_D and η_G are the learning rate.

Nevertheless, finding a suitable clipping bound C usually requires an intensive hyperparameter search [26]. However, Chen et al. [27] pointed out that when training networks with standard loss functions, the gradient norm is typically much larger than a suitable choice of C. As a result, the clipping significantly corrupts the gradient information. The quality of generated data will be greatly impacted, especially, for GANs with an unstable training process. In *IoTSL*, the quality of generated data determines the effectiveness of pretraining. Therefore, we use the Wasserstein-1 metric [28] as the loss, which measures the statistical distance between the real and generated data distributions. Importantly, the Wasserstein distance can be used to generate bounded gradients (gradient norm close to 1) to conform to a suitable gradient bound. To obtain the optimal C, we abandon the hyperparameter search and instead incorporate it into the GAN model's training objective as a gradient penalty term [28]. Finally, by combining the label information y_i of the training samples, we summarize the loss function of the generator and discriminator as follows:

$$\begin{aligned} \mathcal{L}_D = {} & \mathbb{E}_{z\sim p_z(z)}\left[D(G(z|y))\right] - \mathbb{E}_{x\sim p_{\text{data}}(x)}\left[(D(x|y))\right] \\ & + \lambda \mathbb{E}\left[(||\nabla D((\alpha x + (1-\alpha)G(z))|y)||_2 - 1)^2\right] \end{aligned} \tag{6}$$

$$\mathcal{L}_G = -\mathbb{E}_{z\sim p_z(z)}\left[D(G(z|y))\right] \tag{7}$$

where λ is the hyperparameter for the gradient penalty term. The variable $\alpha \sim \mathcal{U}[0, 1]$, uniformly sampled from [0, 1], regulates the interpolation between real and generated samples.

Since $||\boldsymbol{g}||_2 \approx 1$, we obtain the optimal clipping bound $C = 1$. we use the above loss function and the weight update method of (5) to train the GAN model. Finally, we obtain the generator satisfying differential privacy.

In summary, we deployed multiple modules for the system as described above. And we summarize the entire process of *IoTSL* in Algorithm 1. Furthermore, we define the non-iid data imputation module as a function to determine whether the device is a qualified participant and illustrate it in Algorithm 2.

V. Evaluations

In this section, we deploy *IoTSL* on real-world IoT devices and evaluate the performance of every module and overall performance.

A. Experimental Settings

We implemented an *IoTSL* on a real-world edge computing testbed as shown in Fig. 8. We used five identical Raspberry Pi as IoT devices in the testbed. We used a plug-in power meter for measuring the power consumption during training. We simulated the cloud server using a server equipped with an NVIDIA 3090 GPU and an AMD Ryzen 9 5950x CPU. We use the popular VGG-16 [29] as the evaluation model. VGG-16 is a classical vanilla model with 13 convolutional layers; we deploy the first two layers on the devices and the remaining 11 layers on the server. We built the DNN model in Python and installed the Python version on Raspberry Pi to export the model to the embedded platform. We use Python's *socket* module to implement data interaction between the server and the devices. The *socket* module enables reliable TCP/IP communication between two processes on the network. To avoid sticky packets,[1] the sender must first inform the receiver of the size of the data to be transmitted, and then transmit the data after receiving confirmation from the receiver.

B. Data Sets

We use MNIST [16] and Fashion-MNIST [30] as the data sets for evaluation. MNIST contains 70 000 handwritten digital images, and labels, 60 000 of which are used for training and

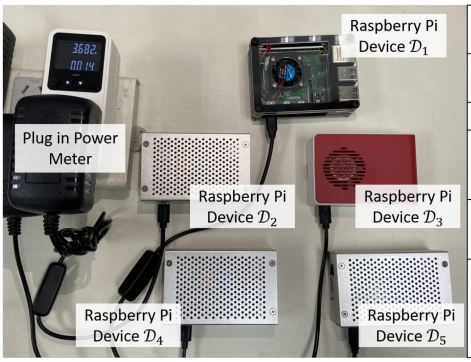
[1] *Sticky Packets:* A common communication problem caused by the TCP/IP protocol will package some small size data which have a short transmission interval to one packet to send.

Algorithm 1: Overall Training Procedure of *IoTSL*

```
/* Pre-training data generated (with
   privacy protection)                          */
Input: The local data sample X_i, ground truth labels Y_i for
       device D_i
Output: Pre-training dataset 𝔻
DeviceExecutes:
for device D_1: device D_n do
    According to Eg. (6) and Eg. (7), {X_i, Y_i} → Generator G_i
    D_i sends G_i → the server
end
ServerExecutes:
Receive G = (G_1, G_2, ..., G_n) from all devices
for G_1 : G_n do
    Generate data based on the local sample distribution of X_i
end
Aggregate generated data → dataset 𝔻
/* Global model pre-training                    */
Input: Dataset 𝔻, training iterations T, batch size B
Output: Pre-training parameters θ_pre,a (model part f'_a) and
        θ_pre,b (model part f_b)
ServerExecutes():
Initialize the parameters θ_pre,a and θ_pre,b
for t in {1,...,T} do
    Sample batch x_i^B ⊆ 𝔻;
    Update θ_pre,a, θ_pre,b ← Loss
end
Send the parameters θ_pre,a → the first device D_1
/* Non-iid data imputation                      */
for device D_1: device D_n do
    DataImputation()(Algorithm 2)
end
/* Global model fine-tuning                     */
Input: Training Round R, training epoch E, The local data
       sample X_i, ground truth labels Y_i for device D_i, the
       pre-training parameters θ_pre,a and θ_pre,b
Output: Parameters θ_a (model part f_a) and θ_b (f_b)
if r = 1 and e = 1 (First epoch of the first device D_1) then
    Initialize the parameters θ_a, θ_b with the pre-training
    parameters θ_pre,a, θ_pre,b
    Update θ_a and θ_b ← local data {X_1, Y_1}
    Send θ_a → device D_2
end
else
    while r ≤ R do
        for device D_1: device D_n do
            Receive parameters θ_a from last device D_{i-1}
            Update θ_a and θ_b ← local data {X_i, Y_i}
            Send θ_a → next device D_{i+1}
        end
    end
end
```

Algorithm 2: Non-iid Data Imputation

```
DataImputation():
Input: Dataset 𝔻, generators G = (G_1, G_2, ..., G_n)
ServerExecutes:
Count the classes of pre-training dataset 𝔻 → L_𝔻
Count the classes of generated data for
G = (G_1, G_2, ..., G_n) → L_1, L_2, ..., L_n
Count missing classes for D_i: L'_i = L_𝔻 − L_i
if L'_i = 0 then
    Not to be imputed // D_i has data for all
        classes
end
else
    Need to be imputed // D_i is defective device
    for L_1 : L_n do
        if L_j ⊇ L'_i then
            Device D_j is the paired member of D_i
        end
    end
end
Send G_j → device D_i
DeviceExecutes:
Generated missing classes data by G_j and mixed with the local
data, the generated quantity is the average of each class in D_i's
local data to prevent class imbalance.
```



Raspberry Pi 4b Specifications	
CPU	Broadcom BCM2711
Memory	4GB DDR4
GPU	500MHz VideoCore VI
Wired Network	Gigabit Ethernet
Wireless Network	802.11ac(2.4/5GHz), Bluetooth 5.0

Fig. 8. IoT devices used in the evaluation.

10 000 for testing, and is divided into ten classes. Fashion-MNIST is an improved version of MNIST with the same data volume size, but the data content has been replaced with frontal images of various consumer products. Both data sets are widely used in the perceptual evaluation of embedded devices since they are 28 × 28 grayscale images with smaller sizes.

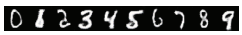
C. Complexity

The first two convolutional layers of VGG-16 are deployed on the IoT devices. Both convolutional layers have 64 convolutional kernels with width 3, and the total number of model parameters is 3.78×10^4. For the input image size $1 \times 28 \times 28$ used in the above data set, the device computation FLOPs (floating point operations) is 29.8M, and the memory required is only 1.2 MB. In contrast, for the common VGG-16 model with 1.34×10^8 parameters, the FLOPs required for computation is 3.03G, and the memory required is 11.24 MB. The models used in *IoTSL* only require about 1% computation, and 10.6% storage compared to the commonly used VGG-16 model. Where FLOPs represent the number of computations and are frequently used as a metric of the model's time complexity.

D. GANs Settings

1) Structure of the Generator: The input vector is a 10-D noise vector drawn from the Bernoulli distribution and is spliced with the sample labels. The vector is then fed into a fully connected layer with $256 \times 4 \times 4$ channels and reshaped into a feature map with 256 channels of size 4×4 using ReLU activation and pixel-norm normalization operations. This feature map is then deconvolved using two consecutive 2-D

TABLE II
EXAMPLES OF GENERATED IMAGES FOR EACH DEVICE (MNIST AND FASHION-MNIST)

	MNIST	Fashion-MNIST
Device $\mathcal{D}_1$		
Device $\mathcal{D}_2$		
Device $\mathcal{D}_3$		
Device $\mathcal{D}_4$		
Device $\mathcal{D}_5$		

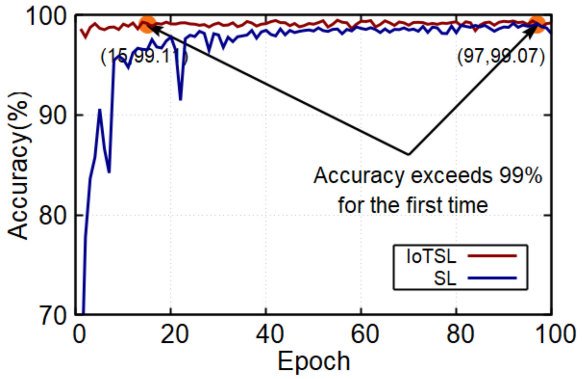


Fig. 9. Comparison of testing accuracy for MNIST.

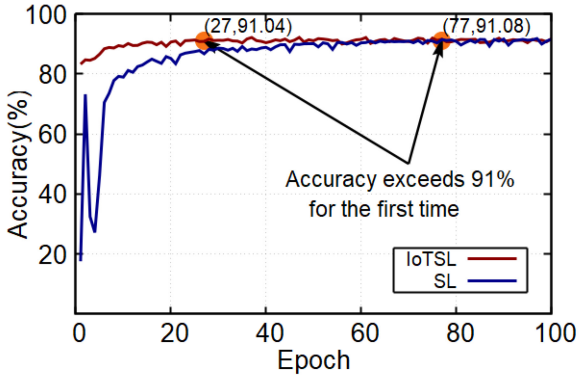


Fig. 10. Comparison of testing accuracy for Fashion-MNIST.

deconvolution layers (128 channels and 64 channels) with kernel size 5 and stride 1. After each layer, the output is activated and normalized. We crop the feature map to 7×7 after the first layer of deconvolution. Finally, a deconvolution layer with a kernel size of 8, a stride of 2, and a channel of 1 that is activated by the sigmoid was used to generate grayscale images with a size of 28×28.

2) Structure of the Discriminator: The discriminator is composed of three convolution layers, each with a kernel size of 5, a stride of 2, and channels of [64, 128, 256], then spectral normalization [31] and ReLU activation. Zero-padding is applied before the second convolutional filter to down-scale feature maps properly in the subsequent convolutions. Finally, the feature maps are flattened and fed into a fully connected layer with 1-D output followed by spectral normalization and linear activation.

Table II shows some examples of the generated images for each device.

E. Efficiency of Global Model Pretraining Module

We evaluate the reduction of *IoTSL* on communication cost. First, we distribute 60 000 images equally among each Raspberry Pi as local data and train the corresponding generator. Next, the generators are sent to the server, where images are generated and combined to produce the data set $\mathbb{D}$. After pretraining the model with $\mathbb{D}$, we send $\boldsymbol{\theta}_{\text{pre},a}$ to the first participating Raspberry Pi $\mathcal{D}_1$. Finally, we fine-tune the model using the device's local data. We conduct a detailed evaluation in the following metrics, and the baseline used for comparison is the conventional SL, which has the same model structure and training data as *IoTSL*.

1) Training Time: IoTSL aims to reduce the communication cost by accelerating the convergence of model training, so we first compare it with the baseline in terms of model training time. Figs. 9 and 10 show the performance of the two approaches in terms of the time of model training, where Fig. 9 shows the MNIST data set and Fig. 10 shows the Fashion-MNIST data set. As shown in the figures, the x-axis represents the training epoch and the y-axis represents the testing accuracy. Both *IoTSL* and SL achieve close to the same final accuracy, proving that *IoTSL* has no negative effect on the performance of the model. Obviously, the convergence of the model becomes faster under the *IoTSL* framework. In the MNIST data set, the accuracy of *IoTSL* exceeds 99% for the first time at the 15th epoch and becomes stabilized, and the total training time is 1.96 h. In contrast, the baseline takes 12.28 h to train and does not surpass 99% until the 97th epoch. In the Fashion-MNIST data set, the accuracy of *IoTSL* exceeds 91% for the first time at the 23rd epoch and stabilizes, and the total training time is 3.01 h. In contrast, the baseline exceeds 91% for the first time at the 77th epoch, and the training time is 10.12 h. As indicated by these results, *IoTSL* effectively increases the model training efficiency and reduces training time.

2) Communication Overhead: To more intuitively verify the effectiveness of *IoTSL* in improving communication efficiency, we compare the number of actual parameters transferred in these two training methods, as shown in Table I.

Due to the model structure of VGG16 and experimental settings, the size of the smashed data $F_i = 64 \times 14 \times 14$ (the input image has 28 pixels, the second convolutional layer of VGG16 has 64 convolutional kernels and a pooling layer), the number of model parameters on each Raspberry Pi $w_i = 37\,824$ and the amount of data on each Raspberry Pi $N_i = 12\,000$. So the number of parameters transferred in one epoch $C_{epoch} = 12\,000 \times (2 \times 64 \times 14 \times 14 + 1) + 37\,824 = 3.01 \times 10^8$. According to the above training time, for the MNIST, the baseline transmits about 2.92×10^{10} parameters until the model converges, whereas *IoTSL* only needs to transmit about 4.52×10^9 parameters, achieving a reduction of approximately 84.5%. For the Fashion-MNIST, the baseline

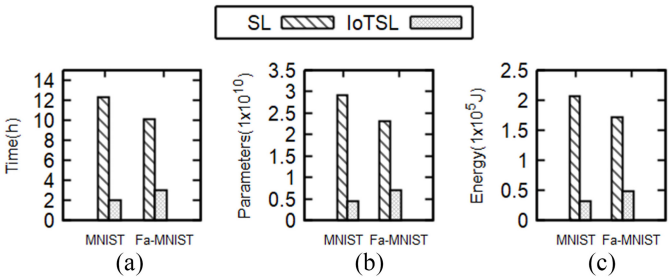


Fig. 11. Computation and communication overhead comparison of SL and *IoTSL*. (a) Training time. (b) Communication overhead. (c) Power.

needs to transmit about 2.32×10^{10} parameters, whereas *IoTSL* only needs to transmit about 6.93×10^9 parameters, achieving a reduction of approximately 70.3%. As a result, *IoTSL* significantly reduces training-related communication overhead.

3) Power Consumption: Energy consumption is crucial for IoT devices. As shown in Fig. 8, we used a plug-in power meter to measure the power consumption of the Raspberry Pi. In *IoTSL*, each participating device takes turns with the server for distributed training, leaving the other devices idle while one device is training. Here, we only measure the power consumption of devices in the training state and summed it up to get the total power consumption. For the MNIST, the total training power consumption of the five devices in *IoTSL* is 3.17×10^4 J, while the baseline requires 2.07×10^5 J. For the Fashion-MNIST, the total training energy consumption in *IoTSL* is 4.91×10^4 J, while the baseline requires 1.71×10^5 J. It can be seen that the improvement of *IoTSL* in terms of device energy consumption is also significant compared with the baseline.

We show the comparison between *IoTSL* and SL using all the above three metrics in Fig. 11. As can be seen from the figure, *IoTSL* outperforms SL in each of the three metrics.

F. Efficiency of Non-IID Data Imputation Module

To evaluate the improvement effect of *IoTSL* on catastrophic forgetting, we first perform non-iid setting on the local data of five Raspberry Pi. There are 6000 images in each of the ten classes in the training data set. Each class's training data is divided into five data blocks and each of which contains 1200 images. We send the data blocks to each Raspberry Pi but only once per data block, to ensure that there is no duplicate local data between devices. In addition, to evaluate the efficiency of *IoTSL* at various levels of non-iid, we employ the four data allocation methods listed below.

1) *Non-IID, Each Device Has Two Classes:* Each device receives two data blocks of varying classes, and each device has different classes of local data.
2) *Non-IID, Each Device Has Four Classes:* Each device receives four data blocks of varying classes, but the classes assigned to each device are random.
3) *Non-IID, Each Device Has Six Classes:* Each device receives six data blocks of varying classes, but the classes assigned to each device are random.
4) *Non-IID, Each Device Has Eight Classes:* Each device receives eight data blocks of varying classes, but the classes assigned to each device are random.

In order to evaluate the four data allocation methods above, we conduct four sets of comparison experiments using SL and *IoTSL*. Fig. 12(a) and (b) shows the testing accuracy for the two data sets using SL, respectively. For the relatively simple MNIST, the highest accuracy under the four non-iid settings are 15.92%, 48.34%, 96.5%, and 99.11%, respectively. For the relatively complex Fashion-MNIST, the highest accuracy are 14.97%, 45.99%, 64.04%, and 86.82%, respectively. As can be seen from the figures, the yellow curves represent the extreme non-iid with only two image classes on each device, where SL fails completely and the testing accuracy is only around 10%. As the degree of non-iid decreases, the learning efficiency gradually improves. It is worth noting that, the testing accuracy shows some regular fluctuations, which is dues to the fact that the training process of SL is alternately and cyclically engaged by the participating devices, and when the data quality on a particular device is better, it has better testing accuracy within its training epoch, and vice versa. The above results show that when the training data is non-iid, the catastrophic forgetting phenomenon has a significant impact on normal model training for SL.

Next, we use *IoTSL* framework to train the generator for each device and send it to the server. Based on the logs, the server pairs generators for the defective devices and finally assembles the entire local data set on the defective devices. In particular, for non-iid one device two classes, where each device has only two classes of images, it needs to receive generator models for the other four devices. To avoid the influence of other factors on the model accuracy, we do not pretrain the model in the preparation phase. Fig. 12(c) and (d) shows the testing accuracy of the two data sets in training with *IoTSL*, respectively. For MNIST, the highest accuracy under the four non-iid settings is 97.27%, 98.13%, 99.02%, and 99.39%, respectively. For Fashion-MNIST, the highest accuracy is 78.42%, 82%, 85.99%, and 89.96%, respectively. In Section V-A, the local data of each device is iid, and the highest accuracy of the two data sets is 99.48% and 91.96%, respectively. The training effect is greatly improved and has close accuracy with iid local data when using the *IoTSL* framework. As a result, *IoTSL* can effectively address the catastrophic forgetting problem of IoT devices caused by a class imbalance in SL.

G. Effect of the Privacy Protection Module on Accuracy

To further reduce the risk of privacy leakage, *IoTSL* combines GANs with a privacy protection module. Even though [27] has demonstrated the efficacy of this approach in terms of privacy protection. However, since the mechanism $\mathcal{M}_{\sigma,C}$ sanitizes the gradients, it inevitably shifts the generated data farther from the real data. In *IoTSL*, we use generated data to pretrain the model, and if the quality of generated data becomes poor due to noise injection, it may have a direct impact on the model training performance.

To evaluate the impact of generated data on training after privacy protection, as a comparison we train GANs without the privacy protection module on all devices with the same model structure of the generator and discriminator and

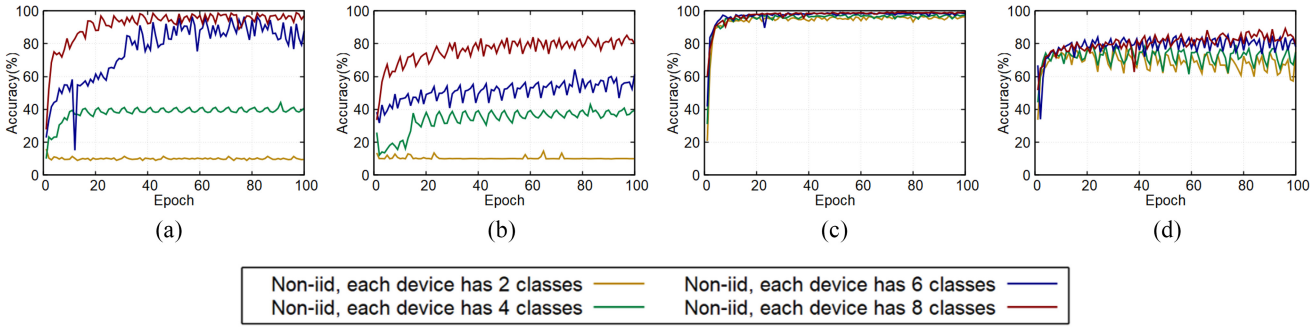


Fig. 12. Comparison of testing accuracy before and after adding the non-iid data imputation module. (a) Testing accuracy of Non-iid MNIST. (b) Testing accuracy of Non-iid Fashion-MNIST. (c) Testing accuracy of imputing MNIST. (d) Testing accuracy of imputing Fashion-MNIST.

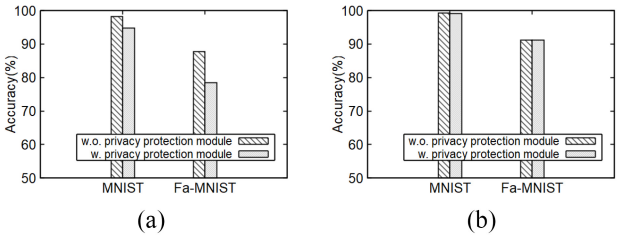


Fig. 13. Effect of the privacy protection module on the accuracy. (a) Comparison of pretraining accuracy with or without privacy protection module. (b) Comparison of fine-tuning accuracy with or without privacy protection module.

parameter settings as before. The data set $\mathbb{D}$ was then generated in the same way, and the model was pretrained. To compare the differences between the generated data and the real data, we use the real data as the testing set. We designed experiments on the two data sets mentioned above, and the results are shown in Fig. 13(a). GANs without a privacy protection module has higher accuracy, 87.78%, and 98.23% in Fashion-MNIST and MNIST, respectively, which is a reasonable phenomenon because the data generated without the added mechanism $\mathcal{M}_{\sigma,C}$ would be closer to the real data. The accuracy of GANs with the privacy protection module is still within acceptable limits, at 78.48% and 94.74%, respectively. Furthermore, we continue to fine-tune the two pretrained models using real data, and Fig. 13(b) shows that their final accuracy is nearly identical, indicating that the privacy protection module has no negative impact on the final accuracy of *IoTSL* while protecting the privacy of local data.

H. Overall Efficiency of IoTSL

To evaluate the overall improvement effect of *IoTSL*, we design the following experiments to compare the training results of *IoTSL* and SL. In practice, the local data distribution of IoT devices is typically irregular. To simulate this scenario, we generate random data distribution for each device, including the classes and the number of data within each class. We send the training data sets to each device based on the random data distribution and train the models with the SL and *IoTSL* frameworks, respectively. In addition, to better evaluate the performance, we design three sets of data distributions (random distributions A, B, C). We used the same model structure and testing data as in previous experiments. Fig. 14(a) and (b) shows the testing accuracy of the two data sets in the SL framework, respectively. As shown in the figure, the testing accuracy rises slowly and fluctuates, and the final results are generally lower. Fig. 14(c) and (d) shows the testing accuracy of the two data sets in the *IoTSL* framework, respectively. In contrast, the convergence of the model is fast, the fluctuation of accuracy is smaller, and the final accuracy is also significantly higher than SL, which shows the effectiveness of the proposed *IoTSL*.

Furthermore, we extend *IoTSL* to the CelebA [32] face data set to evaluate *IoTSL*'s overall performance in more complex tasks. CelebA consists of over 200 000 RGB color face images with 40 feature labels. To create a classification task, we chose three labels: sex, whether smiling, and whether black hair and set the size of each image to 128 × 128 pixels. In comparison to the one-channel grayscale image in the MNIST series data set, the images in the CelebA data set are three-channel color images that are also much larger in size than MNIST. We divided the data set into eight classes based on the combination of the three features mentioned above: {Smiling, Black Hair, Male}, {Not-Smiling, Black Hair, Male}, {Smiling, Not-Black Hair, Male}, {Not-Smiling, Not-Black Hair, Male}, {Smiling, Black Hair, Female}, {Not-Smiling, Black Hair, Female}, {Smiling, Not-Black Hair, Female}, {Not-Smiling, Not-Black Hair, Female}. We chose 10 000 images at random for each class, with 9000 images used for training and 1000 images used for testing. We distributed 72 000 images as local data to IoT devices and 8000 images as the testing data set. We designed three different sets of data distributions (distributions A, B, and C), including the classes and the number of data within each class. Among them, distributions A and B are non-iid and they are used to evaluate the efficiency of *IoTSL* in improving catastrophic forgetting, and distribution C is iid and it is used to evaluate the efficiency of *IoTSL* in reducing the communication costs. Table III shows some examples of generated images for each device with the random distribution A, where the blank part represents the inability to generate images of that class, i.e., there is no data of that class in that device's local data. According to the generated image examples, the local data distribution is completely different for each device. For non-iid imputation, device $\mathcal{D}_1$ and device $\mathcal{D}_5$ are paired with each other, $\mathcal{D}_2$ and $\mathcal{D}_3$ are paired with each other, while $\mathcal{D}_4$ requires generators from multiple devices. Fig. 15 shows the accuracy comparison between using *IoTSL* and SL under different distributions. In distributions A and B, SL causes

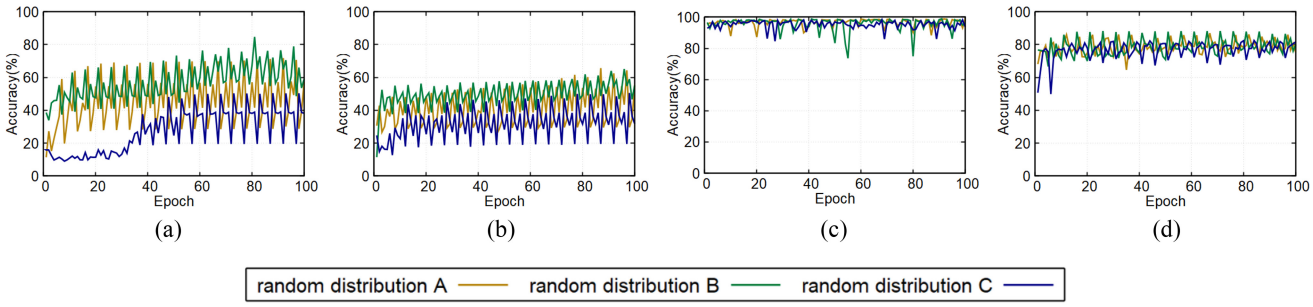


Fig. 14. Overall performance comparison of *IoTSL* and SL. (a) Testing accuracy of SL (MNIST). (b) Testing accuracy of SL (Fashion-MNIST). (c) Testing accuracy of IoTSL (MNIST). (d) Testing accuracy of IoTSL (Fashion-MNIST).

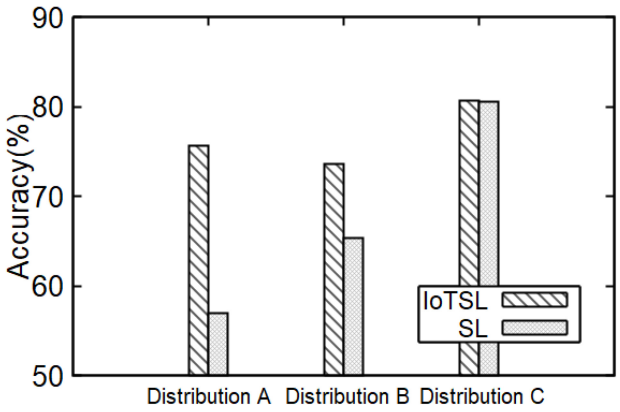
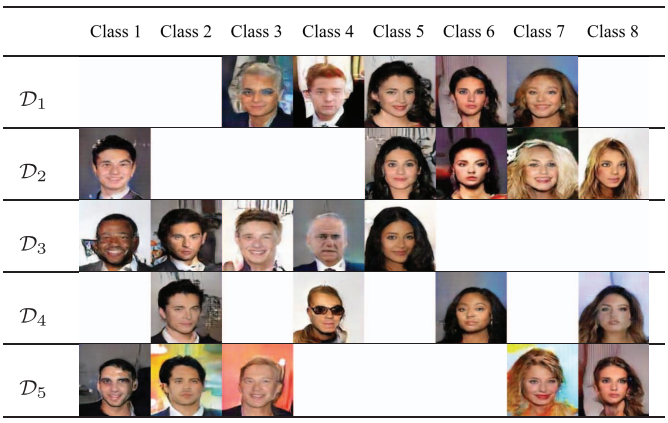


Fig. 15. Comparison of testing accuracy for CelebA.

TABLE III
EXAMPLES OF GENERATED IMAGES FOR EACH DEVICE (CELEBA)



severe catastrophic forgetting, whereas *IoTSL* effectively alleviates this phenomenon and significantly improves accuracy. In distribution C, *IoTSL* and SL obtain close accuracy, but *IoTSL* achieves more than 80% accuracy at the 67th epoch, while SL requires the 652nd epoch. As a result, the above experimental results demonstrate the effectiveness of *IoTSL* in such tasks.

VI. RELATED WORK

Distributed machine learning has long been a hot research topic, particularly the FL framework proposed in recent years, which has been widely used in a variety of specific scenarios [33], [34]. However, many challenges remain when deploying FL in IoT devices, including privacy, communication costs, decentralization, etc. Zheng et al. [35] proposed a system design, which allows clients to only provide obscured model updates while a cloud server can still perform the aggregation, thus, further protecting privacy. Liu et al. [4] proposed a distributed approximate Newton-type algorithm with fast convergence speed to alleviate the problem of federated edge learning resource (in terms of communication resources) constraints. Yang et al. [36] proposed LFNL, a decentralized brain-inspired computing method based on SNNs, enabling multiple edge devices to collaboratively train a global neuromorphic model without a fixed central coordinator.

The serial FL framework (SL) splits the global model, which allows the device side to be assigned a small part of the global model, so it is theoretically more suitable for IoT scenarios. Poirot et al. [37] used SL in healthcare to address the shortage of labeled data from a single data source using collaborative deep learning. Vepakomma et al. [38] similarly used SL in smart health, aiming to avoid sharing raw patient data with the central server to protect privacy. Gao et al. [39] has conducted extensive experiments in real IoT scenarios to compare the performance differences between parallel FL and SL. It emphasizes that SL requires more overall communication, training time, and power overhead in IoT scenarios and that SL is sensitive to non-iid data. Regarding improvements to SL, recent research has used parallel computing to overlap communication and computation time in order to accelerate SL training [40], [41]. For the communication cost, Chen et al. [42] proposed a loss-based asynchronous training scheme to reduce the transmission frequency between client and server, and quantify the transmitted parameters, however, it reduces the accuracy of the global model. For the catastrophic forgetting phenomenon, Kirkpatrick et al. [43] proposed to use elastic weight consolidation to limit the updates on previous clients, Qu et al. [44] applied deep generative replay to mimic data from previous clients or tasks. Unlike the previous work, this article focuses on the communication overhead and sensitivity to non-iid training data of the SL framework and proposes a distributed framework that is more suitable for IoT scenarios. The proposed *IoTSL* framework is unique in that it reduces not only communication costs, training time, and power overhead but also the catastrophic forgetting phenomenon.

VII. Limitations and Future Work

IoTSL is an efficient distributed learning framework for IoT devices. *IoTSL* uses the generative data to pretrain the global model and uses the generative data to impute the device's local data. Due to the training of generators, additional computational costs are inevitably incurred. However, it is a one-time training cost and can be completed offline. In addition, for hardware-constrained IoT devices, the generator training can be deployed on trusted edge devices.

Despite the above limitations, *IoTSL* still has multiple advantages that can be applied to different IoT application scenarios in practice. Future extensions of *IoTSL* can focus on further reducing the training cost of GANs. Some enlightening and pioneering works include [45], which proposes GAN compression for reducing GAN inference time and computational cost. Ren et al. [46] proposed OMGD, which aims to optimize and compress GANs during training, resulting in better image results and reduced computational cost. As future directions to explore, combining these effective lightweight GANs methods in *IoTSL* has the potential to better accommodate different hardware-constrained IoT devices.

VIII. Conclusion

In this work, we propose *IoTSL*, a distributed machine learning framework based on SL that is efficient for resource-constrained IoT devices. *IoTSL* has all advantages of SL and specifically, *IoTSL* has two additional unique advantages. First, *IoTSL* effectively reduces the communication costs between devices and the cloud servers during training. We evaluated using several data sets, and *IoTSL* all significantly reduced communication costs. Second, *IoTSL* effectively addresses the catastrophic forgetting problem that occurs during the training process. Furthermore, *IoTSL* protects the privacy of local data on IoT devices. We have deployed *IoTSL* on real IoT devices and designed extensive evaluations, which show that *IoTSL* achieves improved performance over conventional methods.

Acknowledgment

The authors would like to thank anonymous reviewers for their valuable comments.

References

[1] J. Mao, S. Zhu, X. Dai, Q. Lin, and J. Liu, "Watchdog: Detecting ultrasonic-based inaudible voice attacks to smart home systems," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8025–8035, Sep. 2020.

[2] Z. Yang, B. Liang, and W. Ji, "An intelligent end–edge–cloud architecture for visual IoT-assisted healthcare systems," *IEEE Internet Things J.*, vol. 8, no. 23, pp. 16779–16786, Dec. 2021.

[3] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Comput. Surveys*, vol. 52, no. 6, pp. 1–36, 2019.

[4] Y. Liu, Y. Zhu, and J. J. Q. James, "Resource-constrained federated learning with heterogeneous data: Formulation and analysis," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 5, pp. 3166–3178, Sep.–Oct. 2022.

[5] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016.

[6] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "AutoCompress: An automatic DNN structured pruning framework for ultra-high compression rates," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, vol. 34, 2020, pp. 4876–4883.

[7] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," in *Proc. 32nd AAAI Conf. Artif. Intell. (AAAI)*, 2018, pp. 4596–4604.

[8] Q. Wu, X. Chen, Z. Zhou, and J. Zhang, "FedHome: Cloud-edge based personalized federated learning for in-home health monitoring," *IEEE Trans. Mobile Comput.*, vol. 21, no. 8, pp. 2818–2832, Aug. 2022.

[9] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*.

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

[11] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, Aug. 2018.

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

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

[14] L. Qu et al., "Rethinking architecture design for tackling data heterogeneity in federated learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 10061–10071.

[15] X. Feng et al., "Time-constrained ensemble sensing with heterogeneous IoT devices in intelligent transportation systems," *IEEE Trans. Intell. Transp. Syst.*, early access, May 6, 2022, doi: 10.1109/TITS.2022.3170028.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[17] S. Thrun and L. Pratt, *Learning to Learn: Introduction and Overview*. Boston, MA, USA: Springer, 1998, pp. 3–17.

[18] X. Han et al., "Pre-trained models: Past, present and future," *AI Open*, vol. 2, pp. 225–250, Jun. 2021.

[19] I. Goodfellow et al., "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 27, 2014, pp. 1–9.

[20] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.

[21] R. Prenger, R. Valle, and B. Catanzaro, "WaveGlow: A flow-based generative network for speech synthesis," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, 2019, pp. 3617–3621.

[22] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014, *arXiv:1411.1784*.

[23] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 1322–1333.

[24] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 603–618.

[25] P. Cuff and L. Yu, "Differential privacy as a mutual information constraint," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 43–54.

[26] M. Abadi et al., "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 308–318.

[27] D. Chen, T. Orekondy, and M. Fritz, "GS-WGAN: A gradient-sanitized approach for learning differentially private generators," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 12673–12684.

[28] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 5769–5779.

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

[30] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.

[31] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," 2018, *arXiv:1802.05957*.

[32] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1–9.

[33] Y. Li, K. Hwang, K. Shuai, Z. Li, and A. Zomaya, "Federated clouds for efficient multitasking in distributed artificial intelligence applications," *IEEE Trans. Cloud Comput.*, early access, Jun. 24, 2022, doi: 10.1109/TCC.2022.3184157.

[34] Z. Su et al., "Secure and efficient federated learning for smart grid with edge-cloud collaboration," *IEEE Trans. Ind. Informat.*, vol. 18, no. 2, pp. 1333–1344, Feb. 2022.

[35] Y. Zheng, S. Lai, Y. Liu, X. Yuan, X. Yi, and C. Wang, "Aggregation service for federated learning: An efficient, secure, and more resilient realization," *IEEE Trans. Dependable Secure Comput.*, early access, Jan. 27, 2022, doi: 10.1109/TDSC.2022.3146448.
[36] H. Yang et al., "Lead federated neuromorphic learning for wireless edge artificial intelligence," *Nat. Commun.*, vol. 13, no. 1, pp. 1–12, 2022.
[37] M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta, and R. Raskar, "Split learning for collaborative deep learning in healthcare," 2019, *arXiv:1912.12115*.
[38] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," 2018, *arXiv:1812.00564*.
[39] Y. Gao et al., "End-to-end evaluation of federated learning and split learning for Internet of Things," 2020, *arXiv:2003.13376*.
[40] S. Wang, A. Pi, X. Zhou, J. Wang, and C.-Z. Xu, "Overlapping communication with computation in parameter server for scalable DL training," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 9, pp. 2144–2159, Sep. 2021.
[41] S. Cai et al., "DynaComm: Accelerating distributed CNN training between edges and clouds through dynamic communication scheduling," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 2, pp. 611–625, Feb. 2022.
[42] X. Chen, J. Li, and C. Chakrabarti, "Communication and computation reduction for split learning using asynchronous training," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, 2021, pp. 76–81.
[43] J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci.*, vol. 114, no. 13, pp. 3521–3526, 2017.
[44] L. Qu, N. Balachandar, M. Zhang, and D. Rubin, "Handling data heterogeneity with generative replay in collaborative learning for medical imaging," *Med. Image Anal.*, vol. 78, May 2022, Art. no. 102424.
[45] M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, and S. Han, "GAN compression: Efficient architectures for interactive conditional GANs," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 5284–5294.
[46] Y. Ren, J. Wu, X. Xiao, and J. Yang, "Online multi-granularity distillation for GAN compression," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 6793–6803.

Xingyu Feng received the B.S. degree from Jiangxi University of Finance and Economics, Nanchang, China, in 2017, the M.E. degree from Shenzhen University, Shenzhen, China, in 2020, where he is currently pursuing the Ph.D. degree with the School of Computer and Software.

His primary research interests mainly include Internet of Things, mobile edge computing, and deep learning.

Chengwen Luo (Member, IEEE) received the Ph.D. degree from the School of Computing, National University of Singapore, Singapore, in 2015.

He is currently an Associate Professor with the National Engineering Laboratory for Big Data System Computing Technology and the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include mobile and pervasive computing and security aspects of Internet of Things.

Jiongzhang Chen (Graduate Student Member, IEEE) received the B.S. degree from Shenzhen University, Shenzhen, China in 2020, where he is currently pursuing the master's degree with the School of Computer Science and Software Engineering.

He is supervised by Prof. C. Luo. His current research interests include edge AI, deep learning, and split learning.

Yijing Huang (Graduate Student Member, IEEE) received the B.S. degree from Jiangxi University of Finance and Economics, Nanchang, China, in 2018. He is currently pursuing the master's degree with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China.

His current research interests mainly include Internet of Things and deep reinforcement learning.

Jin Zhang is currently an Associate Researcher with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China.

He worked in CSE, UNSW, and CSIRO as a Research Engineer and the Ph.D. scholarship Holder from 2013 to 2017. During this period, he received the Ph.D. degree as well. He combined AI, machine learning technology with IoT devices, i.e., commercial Wi-Fi laptop or router for smart IoT sensing.

Weitao Xu (Member, IEEE) received the Ph.D. degree from The University of Queensland, Brisbane, QLD, Australia, in 2017.

He is an Assistant Professor with the Department of Computer Science, City University of Hong Kong, Hong Kong. His research generally focuses on IoT, such as smart sensing, IoT security, IoT+AI, and wireless networks.

Jianqiang Li received the B.S. and Ph.D. degrees from the South China University of Technology, Guangzhou, China, in 2003 and 2008, respectively.

He is currently a Professor with the National Engineering Laboratory for Big Data System Computing Technology and the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His major research interests include embedded systems and Internet of Things.

Victor C. M. Leung (Life Fellow, IEEE) is currently a Distinguished Professor of Computer Science and Software Engineering with Shenzhen University, Shenzhen, China. He is also an Emeritus Professor of Electrical and Computer Engineering and the Director of the Laboratory for Wireless Networks and Mobile Systems, University of British Columbia, Vancouver, BC, Canada. His research interests include wireless networks and mobile systems. He has published widely in these areas.

Mr. Leung is serving on the editorial boards of the IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING, IEEE TRANSACTIONS ON CLOUD COMPUTING, IEEE TRANSACTIONS ON COMPUTATIONAL SOCIAL SYSTEMS, IEEE NETWORK, and several other journals. He is a Fellow of the Royal Society of Canada (Academy of Science), the Canadian Academy of Engineering, and the Engineering Institute of Canada.