Evaluation and Optimization of Distributed Machine Learning Techniques for Internet of Things

Yansong Gao, Minki Kim, Chandra Thapa, Alsharif Abuadbba, Zhi Zhang, Seyit Camtepe, Hyoungshick Kim, and Surya Nepal

Abstract—Federated learning (FL) and split learning (SL) are state-of-the-art distributed machine learning techniques to enable machine learning training without accessing raw data on clients or end devices. However, their comparative training performance under real-world resource-restricted Internet of Things (IoT) device settings remains barely studied. This work provides empirical comparisons of FL and SL in real-world IoT settings regarding (i) learning performance with heterogeneous data distributions and (ii) on-device execution overhead. Our analyses in this work demonstrate that the learning performance of SL is better than FL under an imbalanced data distribution but worse than FL under an extreme non-IID data distribution. Recently, FL and SL are combined to form split federated learning (SFL) to leverage each of their benefits (e.g., parallel training of FL and lightweight on-device computation requirement of SL). Our work considers FL, SL, and SFL, and mounts them on Raspberry Pi devices to evaluate their performance, including training time, communication overhead, power consumption, and memory usage with resource-restricted IoT devices. Besides evaluations, we apply two optimizations. First, we generalize SFL by carefully examining the possibility of a hybrid type of model training at the server-side. The generalized SFL merges sequential (dependent) and parallel (independent) processes of model training and thus is beneficial to a system with a large scale of IoT devices, specifically at the server-side operations. Second, we propose pragmatic techniques to substantially reduce the communication overhead by up to four times for the SL and (generalized) SFL.

Index Terms—Split federated learning, split learning, federated learning, distributed machine learning, Internet of Things (IoT)

1 INTRODUCTION

Due to its stunning performance, deep learning (DL) has enabled various applications ranging from image classification, object detection, speech recognition to disease diagnosis, financial fraud detection [1], [2], [3]. One major factor in achieving high accuracy is usually to leverage big data to learn high-level features. The intuitive means is to gather the data centrally and then perform the DL model training. However, data can often be highly private or sensitive. For example, data collected from medical sensors [4] and microphones [5] would be such cases. Consequently, users may resist sharing their data with service/cloud providers to build a DL model. In addition, the data aggregator must pay great attention to the data regulations such as General Data Protection Regulation (GDPR) [6], and California Privacy Rights Act (CPRA) [7]. On the other hand, the centralized data might be mishandled or improperly managed by service providers—e.g., incidentally accessed by unauthorized parties [8], or used for unsolicited analytic, or compromised through the network and system security vulnerabilities—resulting in data breach [9], [10]. Therefore, there is a demand for training DL models without aggregating and accessing sensitive raw data that reside in the client-side [11], [12], [13], [14], [15].

To solve the above problem, distributed machine learning techniques have been developed to train a joint/global model with no direct access to the decentralized local raw data. Such techniques are of great appeal to distributed system applications to reap the benefits from rich data yielded by IoT devices in distributed IoT architectures. Distributed learning techniques keep private data locally (e.g., medical records, voice records, and text inputs) and utilize the data during the learning process to reduce privacy leakage risks. However, there still exists a significant gap in evaluating the
training performance of the techniques in terms of their practicality in the IoT-enabled distributed systems hosting resource-constrained devices.

1.1 Limitations
As most IoT devices are resource-constrained, the DL training inference and training overhead should be first evaluated and then optimized. Currently, most studies focus on the on-device inference performance of resource-constrained devices, assuming that models are trained on resource-rich computing platforms. However, model training on resource-restricted computing platforms is rarely evaluated. This is probably because that training models on resource-restricted IoT devices are still challenging, e.g., to be implemented and managed. Notwithstanding, it is imperative to investigate the training performance of distributed learning techniques as, in most cases, the data yielded by IoT applications (e.g., smart-home and smart-health) are sensitive and should not leave the local devices.

1.2 Our Studies

1.2.1 Learning Performance
This work is firstly to evaluate the training performance of distributed learning techniques under IoT settings with a focus on the popular Federated Learning (FL) [11], [12], [13] and the recent Split Learning (SL) [14], [15]. It is recognized that SL training is sequential, incurring significant time overhead though it has computation advantage at the device side by only training the device-side subnetwork. While FL has to train the entire model on the device, and the training operations happen in parallel. In this context, we consider a new hybrid distributed learning framework, namely splitfed learning (SFL) [16], that explores the respective benefits provided by FL and SL. Nonetheless, current SFL designs [16] are specific, prohibiting their flexible adoption for IoT applications, especially the large-scale IoT devices in the real world. To overcome this limitation, we propose a generalized SFL (SFLG), which is more suitable and flexible for IoT applications as it fits well with large-scale IoT devices.

For each of these distributed learning techniques, we evaluate and compare their learning performance in an end-to-end manner. The learning performance is evaluated with various datasets, and various settings, including (i) independent and identically distributed (IID) data, (ii) imbalanced data, and (iii) non-IID or skewed data reside locally to resemble the heterogeneous data distribution characteristics for IoT devices in the real world. This is the first work to comprehensively compare the learning performance of these promising distributed learning techniques with a concentration on the IoT setting. Moreover, considering the importance of communication efficiency in IoT settings, we have proposed techniques to reduce the communication overhead in SFLG.

1.2.2 On-Device Overhead
Furthermore, there is no empirical study on the end-to-end evaluation of FL, SL, and the SFL on real-world IoT devices, e.g., Raspberry Pi, in terms of their implementation or execution overhead, such as communication cost, power consumption, and training time. Indeed, as highlighted in [17], there is a demand to understand the deep learning performance on resource-constrained IoT/edge device hardware like Raspberry Pi [18]. Experimental results with real-world IoT devices would be useful for practitioners when choosing suitable distributed learning techniques for deployment. Therefore, this work is the first to empirically and systematically evaluate the training of FL, SL as well as SFL in real-world IoT devices. We note that there exist Raspberry Pi implementations on FL [19], [20] but not on other distributed learning techniques. In fact, mounting FL, SL, and SFL on Raspberry Pi is non-trivial, we have summarized the engineering challenges we faced and the corresponding solutions in a manual, which is publicly released for the community.

1.3 Contributions
Overall, this work makes distributed learning more suitable for resource-constrained (e.g., computation and communication are restricted) IoT applications. The main contributions/results (last three) are new or have been renewed in comparison with our conference work [21]) of this work are summarized as follows:

1) We are the first [21] to evaluate SL learning performance in terms of model accuracy and convergence under non-IID and imbalanced data distributions, and then compare it with a popular counterpart FL under the same settings. Our empirical results—up to 100 simulated clients—demonstrate that SL exhibits better learning performance than FL under imbalanced data but worse than FL under (extreme) non-IID data, indicating that SL accuracy is also sensitive to the heterogeneous characteristics of the distributed data. We have also analyzed the rationale behind it. (Section 4)

2) Based on two specific SFL variants, we propose a generalized SFL (i.e., SFLG) for leveraging the advantages of each of SL and FL to complement their shortcomings and then evaluate it with the same settings of FL and SL for a fair comparison. SFLG obviates the cumbersome sequential training process among IoT devices and utilizes the rich resources in the server-side to expedite the training while retaining its low on device computational overhead advantage. (Section 4)

3) We take the first step toward fair comparisons of training performance between FL and SL by mounting both on Raspberry Pi. We provide detailed performance overhead evaluations of training time, amount of memory used, amount of power consumed, communication overhead, peak power, and temperature to serve as a reference for practitioners. In addition, (i) effects of the number of split layers in the SL, (ii) effects of models with different complexities for both SL and FL are quantified and compared. The further IoT implementation of SFL corroborates its substantial training time reduction while still maintaining the low computation overhead inherited from the SL. (Section 5)

4) We validate that SFLG achieves good flexibility to trade-off learning performance, scalability, and training time expeditation when the IoT devices are...
large-scale. In addition, techniques of reducing SL and SFLG communication overhead are proposed and experimentally validated to better fit the scenarios where IoT applications that could be bottlenecked with communication. Source codes of this work are released for facilitating future research and deployment https://github.com/garrisongys/SplitFed.1 (Section 6)

The remainder of this paper is organized as follows: Section 2 details background about distributed learning techniques. Section 3 presents our generalized SFL (namely SFLG). In Section 4, we comprehensively evaluate SL, FL, and SFL under various heterogeneous data distributions for end-to-end comparisons. Section 5 mounts these three distributed learning techniques on Raspberry Pi to empirically evaluate and compare their implementation overhead. Section 6 validates the advantages of SFLG in large-scale IoT devices and further investigates pragmatic techniques to reduce the communication overhead of SL and SFLG. Section 7 concludes this work.

2 BACKGROUND

In this section, we provide a brief background on federated learning, split learning, and splitfed learning.

2.1 Federated Learning

Federated learning (FL) is illustrated in Fig. 1a where we focus on the representative FL learning with the average algorithm FedAvg for local model aggregation [22]. During the training process, the server first initializes the global model $w_0$ and sends it to all participating clients. After receiving the model $w_t$, each client $k$ trains the global model on its local data $\mathcal{S}_k$, $w_k$ is the number of training samples held by client $k$ while $s$ is the total number of training samples across all clients. Afterward, each client returns the locally updated model $w_{t+1}^k$ to the server. The server then aggregates all those models to update the global model $w_{t+1}$. The above process (i.e., one round) is repeated until the global model converges.

According to [22], instead of training the model with local data only one epoch, each client trains the local model for several epochs before sending it to the server in one communication round, which is commonly used for FL optimization to reduce communication overhead. Although FedAvg usually works well, specifically for non-convex problems, there are no convergence guarantees. FL may diverge in practical settings, especially if data are non-IID or distributed in an imbalanced manner.

2.2 Split Learning

Unlike FL, in which each client trains the whole neural network, split learning (SL) [14], [15] divides a neural network-based model into at least two sub-networks and then trains the sub-networks separately, on distributed parties (e.g., client and server). A representative example of SL is illustrated in Fig. 1b, where $C_3$ is the cut layer that divides the whole network into two sub-networks. The first sub-network $h_t^1$ is trained and accessed by the client; the second sub-network $w_t$ is trained and accessed by the server. To train a model, SL has forward and back-propagation in the network. In Fig. 1b, the training is as follows: the client carries forward propagation over the input data and then sends the activations of the cut layer, called smashed data $(A_t^1)$, to the server. The server carries forward propagation over the smashed data and calculates loss. Then, back-propagation starts over the loss and yields the gradients of the smashed data $(\forall \ell(A_t^1; w_t))$ during the process. Afterward, the gradient is sent to the client, which then carries its back-propagation. For the SL training and testing, the server has no access to clients’ sub-networks and data, thus preserving the clients’ data privacy. Besides, each client only needs to train a sub-network usually consisting of a few layers, and most layers reside in the server. Therefore, a client’s computation workload is reduced as the other benefit.

The learning performance (e.g., model accuracy and convergence) of SL has not been investigated yet when the dataset is non-IID or distributed in an imbalanced manner, which we consider in this work.

2.3 Splitfed learning

SL greatly reduces the computation requirement on the client-side as it only computes on a (small) sub-network. However, it needs to sequentially iterate over each client, which results in prolonged training time if multiple clients are present. In FL, each client usually interacts with the server in parallel; thus, training can be completed faster than that of SL. However, each client has to train the entire model that incurs higher computational overhead.

Recently, SL and FL are blended to take advantage of each, which is called splitted learning (SFL) [16]. In SFL, all clients compute independently in parallel, i.e., they send/
receive their smashed data to/from the server in parallel. The client-side sub-network synchronization, i.e., forming the global client-side network, can be done by aggregating (e.g., weighted averaging) all client-side local networks in a separate server, called fed server. Two ways of server-side sub-network synchronization are presented in [16], which correspondingly results in two specific variants of SLF.

- **SFLV1:** The SFLV1 firstly performs parallel and independent training over the smashed data of each client, which results in the number of server-side sub-networks equals the number of clients. Later, all the sub-networks are aggregated (e.g., weighted averaged) to form the global server-side network. This is referred to as SFLV1.

- **SFLV2:** The SFLV2 performs sequential server-side sub-network training over the smashed data of each client—note the client can still send their smashed data concurrently. This keeps only one copy of the sub-network on the server-side, and it is the global server-side network going through all smashed data sequentially. This is referred to as SFLV2.

By observing the possibility of merging these two specific SFLV1 and SFLV2 algorithms, this work generalizes them. The generalized algorithm enables a varying number of server-side sub-networks other than that of SFLV1 and SFLV2 in the main server-side while training. Thus one can flexibly choose the number based on the available server-side computational capacity, which is suitable for large-scale IoT devices focused by our study. The flexibility of the generalized SFL will be validated and discussed in Section 6.1.

### 3 Generalizing SplitFed Learning

In the generalization of SFL, the client-side operations, including client-side model synchronization, are kept the same. The only change is in the server-side training operations. In the generalised SFL, we call SFLG, firstly, the server considers \(|G| = n\) groups, where each group has several clients (here, clients mean clients’ identifications). In a simple setup, the clients in a group are unique, and no two groups share the same client. Then, the server-side model is trained and updated sequentially within the group (only one copy of the server-side model is kept), whereas the group operates independently and in parallel. Thus, at the end of each round, we get the \(|G|\) number of trained server-side submodels. Though this work is presented with the label sharing case, it is possible for various other configurations, such as without label sharing, extended split, and vertically partitioned data, and it is performed similarly to the description provided in split learning [24], [25].

To ease the understanding of the SFLG, we use an example shown in Fig. 2 for description. SFLG generally follows the below steps in each round:

1) At the beginning of the training, all clients receive the same client-side subnetwork \(h\) and the server starts with the server-side subnetwork \(w\). The server chooses the number of groups (this example has two groups) and keeps it the same during the whole training.

2) Once the training starts, all clients perform forward propagation on the client-side subnetwork on their local data in parallel and independently. Then, they send their smashed data (the output of the cut layer) concurrently to the server, as indicated by the label \(\circ\) in Fig. 2.

3) The server assigns all the smashed data coming from one client to one group - the choice of group is random.

4) The forward-backward operations inside a group happen in a sequential manner to the clients’ smashed data, whereas the groups operate in parallel. Each group provides one server-side model.

5) The clients receive the corresponding gradients of their smashed data (obtained at the end of the back propagation on the server-side model, and it is indicated by the labels \(\circ\) and \(\odot\) sequentially in Fig. 2), and then complete their back propagation, resulting in \(h_1, h_2, h_3, h_4\) local client-side models.

6) The server aggregates \(F_{\text{avg}}()\) the server-side subnetworks \(w_1, w_2\) to form \(w\) and also aggregates the device-side subnetworks to form \(h\) after receiving \(h_1, h_2, h_3, h_4\). The global server-side subnetwork and the global client-side subnetwork are \(w\) and \(h\), respectively, which will be used in the next round.

Notably, the aggregation of server-side and device-side subnetworks is performed synchronously at the end of each round. Here, we have simplified the aggregation with a single server. In practice, the server-side and client-side subnetworks can be aggregated separately by the main server and one client server, respectively. The usage of the client server can prevent the main server from accessing the client-side subnetwork, providing an additional privacy benefit.

As for SFLG, it is not difficult to see the following:

**Preposition 1.** (i) If group \(|G| = n\) at the server-side and each group includes a unique client, and no two groups include the same client, then SFLG is SFLV1. (ii) If group \(|G| = 1\) at the server-side and that group includes all the clients, then SFLG is SFLV2.

This preposition will be validated in Section 6. The straightforward benefit of SFLG, including SFLV1 and SFLV2 variants, is power saving, that is, the client-side training can be performed in parallel for the expedition—waiting is no longer required by setting the IoT devices into...
idle status. Moreover, one research challenge—the worse accuracy performance under non-IID that is recognized in our previous study [21]—faced by SL can be mitigated.

4 LEARNING PERFORMANCE EVALUATION AND COMPARISON

4.1 Datasets and Models

Sequential data or time-series data is pervasively collected and processed by IoT devices. For example, people can order and purchase goods using speech commands at their voice assistant. Wearable medical sensors are used to monitor users’ health status in real-time. In this context, we choose two such popular datasets: speech command (SC) and ECG for experimental evaluations, as summarized in Table 1. The SC is a personalized dataset, and the ECG is a medical dataset. Both datasets can be privacy-sensitive, and users are unwilling to share them. Regarding model architecture, the 1D CNN has been recently shown to be efficient for dealing with sequential data [26], which we have adopted. As a matter of fact, the major consideration of 1D CNN usage instead of sequential machine learning models such as LSTM and RNN for sequential data is because there is no effective solution to splitting the sequential model in the SL setting. The SL is currently only applicable to vertical model architectures, in particular, the CNN.

Notably, we have indeed considered image classification tasks and 2D CNN models. Although Raspberry Pi is regarded as a high-end IoT device [27], its computing resources are still quite limited compared with traditional computing devices such as servers and even PCs. We first tried to run MobileNet [28]. When we trained the CIFAR10 dataset [29] with MobileNetv1 (20 conv2D layers with 3,228,170 model parameters in total), it took 8 hours 41 minutes for FL per round with 1 local epoch. For SL, it took about 2.5 hours for one epoch across five Raspberry Pi devices, when only the first two layers are running on the Raspberry Pi device. In addition, we tried to run ResNet20 (20 2D layers and 269,722 parameters). When we ran SL across 5 Raspberry Pi devices, it took about 1 hour to finish one round when the first two layers were only running on the Pi devices. When we also ran FL across 5 Raspberry Pi devices, it took 37 minutes for one round with one local epoch. In this case, we used a PC to serve as a server (specifications of the PC are the same as that detailed in Section 5.1).

Therefore, this work mainly uses two sequential datasets and 1D CNN models, summarized in Table 1, for evaluation and comparison.

4.1.1 Speech Commands (SC)

This task is for speech command recognition. The SC contains many one-second ‘.wav’ audio files: each sample has a single spoken English word [30]. These words are from a small set of commands and are pronounced by a variety of people. In our experiments, we use 10 classes: ‘zero’, ‘one’, ‘two’, ‘three’, ‘four’, ‘five’, ‘six’, ‘seven’, ‘eight’, and ‘nine.’ There are 20,827 samples. 11,360 samples are used for training, and the remaining samples are used for testing.

4.1.2 Electrocardiogram (ECG)

MIT-BIH arrhythmia [31] is a popular dataset for ECG signal classification or arrhythmia diagnosis detection models. Following [32], [33], we collect 26,490 samples in total which represent 5 heartbeat types as classification targets: N (normal beat), L (left bundle branch block), R (right bundle branch block), A (atrial premature contraction), and V (ventricular premature contraction). Half of them are randomly chosen for training, and the rest of them are for testing.

4.2 Data Distribution Considerations

In practice, data is often distributed among clients in an imbalanced manner, e.g., some sensors are more active than others—with more data, and non-IID distributed, e.g., a single person’s data can only be collected [23]. Going forward, we evaluate the learning performance of FL, SL, and SFL under the same data distribution setting, including IID, imbalanced, and non-IID for quantitative comparisons. In our settings, the non-IID is to resemble the worst statistical heterogeneity of the local client data. For example, the extreme ‘one-client-one-class’ case, which means that the each client has only one class data. In contrast, the imbalanced data distribution does not have such an extreme skewness. Each client can still hold samples from all classes but with imbalanced samples per class. In the IID case, we use uniform sampling to ensure each client has balanced data distribution per class, besides data from all classes. So that the non-IID has the worst data distribution heterogeneity while the IID has the best. In all cases, the data distributed to each client are non-overlapped.

As for the SFL, herein we focus on the SFLV1 (that is equal to set |G| = n of the SFLG) as it gains the maximum parallelization capability, which ultimately outsources and reaps the rich computation in the server. This is always preferable, in particular, when the number of participants is not that large. We defer comprehensive learning comparisons among our generalized SFLG with specific SFLV1 (|G| = n of the SFLG) and SFLV2 (|G| = 1 of the SFLG) [16] to demonstrate the SFLG flexibility in large-scaled IoT devices in Section 6.1.

4.3 IID and Balanced Dataset

Starting with ideal IID and balanced data distribution, we evaluate the performance of FL, SL, and SFLV1 with both SC and ECG datasets. For all tests in this section, if there is no explicit statement, the 4conv+2dense 1D CNN model architecture is used. The learning rate is set to be 0.001. The batch size = 32. Moreover, the experiments in this section

| Dataset | # of labels | Input size | # of samples | Model Architecture | Total Parameters | Total Model Accuracy (Centralized data) |
|---------|-------------|------------|--------------|--------------------|------------------|----------------------------------------|
| ECG     | 5           | 128        | 26,490       | 1D CNN             | 66,901           | 97.78%                                 |
| SC       | 10          | 32,187     | 26,490       | 1D CNN             | 522,584          | 85.75%                                 |

2. The source code is adopted from https://github.com/Tshzzz/cifar10.classifier/blob/master/models/mobilenet.py.
3. Since the number of training samples for CIFAR10 is 50,000, we use 5 clients and each client holds 10,000 images for both FL and SL. We run it in default without delicate optimization.
4. Source code is adopted from https://github.com/akamaster/pytorch_resnet_cifar10/blob/master/resnet.py.
are performed with GPU assistance (some experiments are done with Google Colab that provides a free Tesla of 80 GPUs) to expedite simulations of up to 100 users.

Figs. 3 and 4 detail the testing accuracy as a function of the number of rounds when FL, SL, and SFLV1 are trained by a different number of clients—2, 5, 50, and 100 clients. We can see that SL can always converge relatively faster than FL with one local epoch—notably for SL and SFLV1, the local epoch can only be 1. FL struggles with convergence, especially when the number of clients becomes large.

The testing accuracy exhibits a drop after reaching an optimum point (especially the SC dataset in Fig. 4). Thus, an increasing number of rounds appears not to help to improve accuracy. Stopping training at the optimal point saves training time. In addition, SL always exhibits an unstable learning curve with a high number of spikes.

Furthermore, SL’s model accuracy cannot reach the baseline accuracy of the centralized model—85.29% for the SC and 97.78% for the ECG, as detailed in Table 1. This limitation is clearly shown in Fig. 4, when the number of clients is 50 or 100.

These results indicate that the SL model accuracy and convergence performance are not always the same as that of training a model through centralized data. Our findings are consistent with the previous conclusion in [14]. However, we note that our findings are more generalized because we do not assume that the order of the data that arrived at multiple entities should be preserved, and the same initialization is used for assigning weights.

As for the SFLV1, under expectation, its performance approaches FL. This is because, from the model aggregation perspective, the SFLV1 is similar to FL. The only difference is that now the local model is not fully trained on the client but split by the IoT device and the server to reduce the computational overhead on-device.

**Notes:**
(i) The SL learning performance is affected by the number of clients that is consistent with the previous conclusion in [14]. (ii) The SL always outperforms the FL in terms of convergence speed in our experiments. Its training process exhibits unstable spikes, and the testing accuracy turns down once it reaches the optimal. (iii) The SFLV1 performance is close to the FL when the number of local epochs is set to be 1.

### 4.4 Imbalanced Data Distribution

We assume the data are distributed among clients following the normal distribution to simulate the realistic imbalanced data distribution. Larger the sigma/variance, the more imbalanced the data distributed. For example, when the number of clients is 10, and the total number of SC training dataset is 11360, the minimum number of training samples held by one client could be as few as 48, while the maximum number of training samples held by a client could be 3855—this is the setting for Fig. 5d. We simulated clients up to 100. Given the same number of clients, same data distribution is applied to FL and SL and SFLV1.

According to Fig. 5, FL is hard to achieve the baseline accuracy of the centralized model, even when multiple local epochs per round are adopted for a large number of clients. For the SL, its model accuracy also deteriorates when the number of clients is large, e.g., Figs. 5e and 5f. In addition, FL converges slower, especially when the number of clients goes up, e.g., 50 and 100 cases. Usage of more local epochs per round can expedite the convergence issue—but it cannot wholly prevent—given the similar communication overhead. However, more local epochs will proportionally prolong training time and thus consume more power on the IoT device, although it can reduce the communication overhead. SL is less sensitive to imbalanced data distribution since it always demonstrates a faster converge. In Fig. 5f, we can see that the training of SL does not learn for the first 50 rounds/epochs. Once it starts learning, it indeed finds convergence quickly. As for the SFLV1, its learning performance closely approaches FL.

**Notes:** SL learning performance is affected by both the number of clients and imbalanced data distribution. In most cases, SL converges faster than FL in our experiments. The SFLV1 learning performance closely approaches FL.

### 4.5 Non-IID Data Distribution

For the non-IID setting, the SC and ECG datasets are first sorted by class. Each client then receives data partition from only one single class, two classes, three classes, four classes, and five classes.
Results are detailed in Fig. 6. Under extreme non-IID cases (each client has only one class data), both SL and FL face convergence difficulties. FL convergence significantly. Other than 1 client on a class case, the FL can always learn to converge better. For SL, we can see that its learning almost dies. In fact, we found that the predicted label all go to a single class.

Notes: SL is very sensitive to highly skewed data distribution. In our experiments, FL outperforms SL under non-IID data settings, especially in extreme cases. Again, the SFLV1 learning performance under non-IID is approaching FL.

Theoretical Clues: As for FL, the worsened learning performance under statistical divergence or the non-IID data distribution has been well noticed in previous works [34], [35]. The root cause is that the local objective diverges when the local model trains on non-IID data. So that, in this context, when the server aggregates local models, the global model faces convergence hardness. There have been efforts to theoretically guarantee the convergence of FedAvg that is the de facto optimization method in the federated learning conditioned on strong convex and smooth problems [36]. A recent method, namely FedProx extends FedAvg to be more general, tackles both system heterogeneity and statistical heterogeneity in a more efficient manner. Generally, FedProx allows the local model to be updated using customized local settings, e.g., number of local updates, according to its local data statistical characteristics rather than a fixed global setting used in FedAvg. As a consequence, the FedProx demonstrates a significantly more stable and accurate convergence behavior, especially in the non-IID setting.

As for SL, it exhibits worse learning performance than FL under the extreme non-IID setting. We identified that this is a result of a phenomenon termed as “catastrophic forgetting,” when a model is serially trained among clients [37], [38]. In this case, the trained model highly favors the data it has most recently seen. We recognize that the SL shares similarity with the serial training named as cyclic institutional incremental learning [39] (CIIL) except that the SL training uses two sub-networks separately charged by two entities, whereas the former CIIL only utilizes one entity to train the entire network. More specifically, in CIIL, firstly, each institution trains the model and then passes it to the next institution for training until all have trained once. Then this process is repeated, fixing the number of training epochs at each institution and repeatedly cycling through the institutions. As we can see, this process is eventually similar to SL. In this kind of serial training, the trained model highly favors the data that it has most recently seen. This explains that SL cannot converge under extreme non-IID settings, e.g., when a client only holds samples all from a single-class. In this case, the model trained by the next client with different class data appears to completely forget the previous knowledge, thus rendering convergence difficult. However, this forgotten tends to be eliminated once a client can own more classes’ data.

Opposed to SL, even when the number of data samples is extremely uneven in FL, there would still be some contribution retained from each client [40]. Therefore, this explains the better learning performance of FL compared with SL under the extreme non-IID setting.

Notably, imbalanced data distribution is inclusive of the non-IID data distribution. This is because some devices can hold skewed data (majority samples are from few classes) as well. Therefore, the degraded convergence in this imbalanced setting compared with IID data distribution can also be explained.

4.6 Partial Participation

We have previously assumed that all devices participate in the global model update in each round. However, there may be some devices that lose their connections due to unstable networks, usually referring to “stragglers” [35], [36], [41]. We now consider the situation of partial participation. In this context, we set the participation rate to be 20%, 40%, 60%, 80% and 100% (full participation) for SL, FL, and SFLG(50) respectively. The 50 of SFLG means 50 groups are divided out of 100 clients, detailed in Section 6.1. In this context, to the best of our knowledge, we are the first to investigate SL and SFLG learning performance under partial client participation.

Fig. 6. Testing accuracy of FL (1 and 5 local epochs for 1 round), SL, and SFLV1 over rounds with Non-IID dataset setting. (# class(es)) refers that each client is with data from # number of class(es).
The total number of devices is 100, there are a fraction of devices randomly selected in each round. To expedite, we run only 50 rounds (previously, this number is 400) for all three distributed learning techniques (one local update only for FL for simplicity) using the ECG dataset. We have considered both IID and imbalanced data distribution into account. Then we gain the testing accuracy when 50 rounds of training are completed for apple-to-apple comparisons. Experimental results are shown in Fig. 7. A fraction of devices are randomly selected per round to participate in the distributed learning. We can see that partial participants usually do not degrade the training performance for SL, FL, and SFLG in our experiments. Previous studies have shown that the FL can still be converged even in an extremely small participation rate, e.g., 1%—in this extreme participation rate, it does exhibit a slow-down convergence rate [34], [35], [36]. The testing accuracy appears to be better in the FL case when the participation rate goes down to 20%. One potential reason is that each local model in FL has a slightly varying optimization direction. The performance of the global model may be degraded when redundant local models are used in the average operation, as they may cancel with each other to some extent.

We note that the testing accuracy of the SFLG is between the SL and FL. This agrees well with previous experimental results when all clients are participated. Secondly, as for the SL, the testing accuracy of the imbalance distribution is usually worsen than that of the IID distribution. This can be explained by the forgetting phenomenon detailed in Section 4.5 as the imbalanced distribution expedites the forgetting. As for the FL, the testing accuracy of the imbalanced distribution is usually better than that of the IID distribution (this aligns with the results by comparing Figs. 4 and 5 in the first 100 rounds). The reason maybe that a certain level of statistical heterogeneity makes the global model to generalize better, thus converging faster. In addition, the forgetting phenomenon is relatively mitigated in the FL. As for the testing accuracy of the imbalanced and IID distribution, respectively, in the SFLG, its performance again lies between the FL and SL.

As a byproduct, considering the fact that selecting a fraction of devices per round instead of full device participation is usually sufficient to ensure model convergence, such partial device participation can serve one efficient manner to reduce communication overhead as those unselected ones do not need to communicate.

5 IMPLEMENTATION OVERHEAD EVALUATION ON RASPBERRY PI

Using the ECG dataset, we evaluate various overhead metrics such as time, power, communication, and memory when running FL, SL, and SFLV1 on Raspberry Pis that are representative IoT devices to provide a benchmark under real-world IoT settings. In particular, we simulate one typical IoT application scenario, as illustrated in Fig. 8, similar to [42], which can be a smart home setting. According to [27], the IoT device can be generally categorized to high-end IoT device and low-end IoT device. The low-end IoT devices are temperature, motion sensors, and RFID cards, which are usually strictly resource-constraint. They may not even support an OS such as Linux to run a machine learning framework, e.g., PySyft, PyTorch, and TensorFlow. High-end IoT devices are simple devices like Raspberry Pi. Hence, in this simulated IoT application scenario, Pi serves as a gateway, which aggregates data from low-end IoT devices, e.g., sensors, and interacts with the server to perform distributed learning tasks.

5.1 Experimental Setup

We use the Raspberry Pi 3 model BV1.2 (Fig. 9) with the following settings: PyTorch version 1.0.0, OS Raspbian GNU/Linux. Please see the demo for more details (https://www.youtube.com/watch?v=x5mD1_EA2ps).
We have made a unified manual guide of installing Pytorch v1.0.0 on Raspberry Pi at https://github.com/Minki-Kim95/RaspberryPi. We believe that this manual will help developers because we explain how to address the errors during installation, which are non-trivial to resolve, and there are no solutions online during our study.

5.2 Measurement Methods of Performance Metrics

Time Overhead. We use Python’s time library to measure the training time containing the communication time between client and server. We set the time as \( T_{\text{start}} \) when the model starts training. Once the training is finished, we set the time as \( T_{\text{end}} \). As a result, training time is \( T_{\text{end}} - T_{\text{start}} \).

Memory Usage. We use the Linux command `free -h` for measuring the memory usage. This command provides the memory information of `total, used, free, cached` and available. The total memory of the Raspberry Pi device used in this experiment is 926 MB. The focus here is to record and report the used memory during training.

Power Consumption. We use a plug-in powermeter, as shown in Fig. 9, to measure the power consumption. We measured the power consumption in the unit of kilowatt-hour (kWh).

Temperature. We use Python’s CPU TEMPERATURE() function from the CPU TEMPERATURE library to monitor the temperature of the Raspberry Pi CPU. Notably, the device temperature can go high, e.g., \( 80^\circ C \) during training. Therefore, it may be necessary to cool down the device. A cooling fan can be attached to the Raspberry Pi (Fig. 9). This practice can effectively cool it down from \( 83^\circ C \) to \( 54^\circ C \).

Communication Overhead. We measure the transmitted data size from each client to the server and vice versa. We use the PycKle library to monitor the size of the transmitted data. We use the router DGN2200 v4 (N300 Wireless ADSL2 + Modem Router) for wireless communication between Raspberry Pi and the server.

5.3 Evaluation Considerations

We set one local epoch per round for FL and SFLV1 in all experiments. We compare implementation overhead by presetting a fixed number of 100 rounds. We always use the learning rate of 0.001. We considered the following three evaluation settings:

- We first evaluate SL when a different number of layers is partitioned and running on the device, given the same model architecture. Specifically, one, two, three layers are split and running on the device. This shows the advantage of SL to relax the device side computing overhead. (Section 5.4).
- We then evaluate FL and SL across five devices with different model architectures. For the SL, two split layers run on devices regardless of the entire layers of model architectures. This shows the SL is invariant to the model complexity as long as the device sub-network is invariant. (Section 5.5).
- We further evaluate FL, SL, and SFLV1 across a range of devices from two to five with the same model architecture. This shows the SFLV1 on-device computation overhead inherits the advantage of SL, while greatly avoids its undesirable sequential training procedure among devices. (Section 5.6).

For all evaluations, we report the performance overhead for a single Raspberry Pi device because we are interested in the client’s overhead.

5.4 Effects of Number of Split Layers in SL

Experiments are carried on five Raspberry Pi devices to observe the effect of the number of split layers for SL. All effects except the time overhead will apply to the SFLV1, as the latter by design should share the same computation overhead with SL.

Results are detailed in Fig. 10. The communication overhead remains the same regardless of the number of layers at the device-side because the communication overhead in SL hinges on the number of parameters (size of the smashed data) in the cut layer rather than the number of split layers. The memory usage in Fig. 10c, shows only a slight increase with the number of split layers. Most noticeably, time overhead (illustrated in Fig. 10a) and energy overhead (depicted in Fig. 10d) increase with the number of split layers because the model complexity increases for training. Therefore, in practice, from the overhead reduction perspective, it is preferred to run a few layers only at the device-side for SL.

5.5 Effects of Model Complexity

To observe the effects of different models with varying complexity for SL and FL, we perform experiments on five Raspberry Pi devices. The models have a varying number of convolutional layers ranging from four to eight, thus varying model size. For SL, the on-device subnetwork is fixed.
According to the results depicted in Fig. 11, the overhead, including time, communication, memory used, and energy consumed by Raspberry Pi devices (linearly), increases with the model complexity (defined by the number of layers in the model) for FL. In contrast, the overhead remains more or less constant for SL because the number of layers running on each client is fixed. Based on these findings, SL becomes more advantageous when we consider a model with higher complexity.

5.6 A Comparison Among FL, SL, and SFLV1

Here, we evaluate and compare FL, SL, and SFLV1 when the number of Raspberry Pi devices (clients) ranges from two to five. Admittedly, the number of Pis implemented is not high. Therefore, this experiment (i) focuses on the overhead brought to individual Raspberry Pi rather than the server-side, and (ii) demonstrates the SFLV1 qualitative advantage in terms of both computation reduction in comparison with FL and training time reduction in comparison with SL. One can easily extend the number of clients beyond five by adopting the released artifact (source code, user guide, and demo) of our experiment. The used model architecture has four 1D CNN layers and two dense layers. For SL as well as SFLV1, the first two 1D CNN layers run on Raspberry Pi devices. It is worth mentioning that this implementation overhead is newly performed. The peak power under 2W cannot be measured by the powermeter (name: Bplug-S01, manufacturer: a-nine) used in this experiment; this explains the occasionally missing peak power of SLV1 in Fig. 12.

The performance results are presented in Fig. 12. As for the time overhead in Fig. 12a, FL reduces as the number of devices increases. This is due to the decrease in the local data size. SL slightly increases since each device runs the training sequentially. Overall, SL usually takes several times longer than the FL, given the same number of rounds. In terms of the SFLV1, we can see that its computation overheads, including communication, used memory, peak power, and temperature, are all the same as the SL, which are under expectation. Significantly, the training time is greatly reduced in comparison with the SL because now the IoT devices run in parallel. In general, this reduction becomes more obvious given that the number of devices is large. It is worth noting that the server in the experiment is eventually a personal laptop, and there is no multiple-thread optimization made for the current SFLV1 evaluation. If the server is with a cluster of CPU and GPU cores, and the multiple-thread optimization is performed, we expect the training time will be close to the FL by fully exploiting the parallelizing capability.

The communication overhead is presented in Fig. 12b. FL stays relatively constant because the model parameters determine the FL’s communication overhead rather than the local data size. SL and SFLV1 communication overhead

Fig. 11. Overhead performance of FL and SL when the number of layers of the model varies from 4 conv to 8 conv, thus with differing model depth/complexity. All tests were performed with the 100 Gbit/s dedicated LAN. For SL, the first two convolutional layers run at the client-side. For communication overhead, we only show the results for FL because SL’s communication overhead (1054 Mbytes) is not changed as well as significantly greater than FL’s communication overhead, as shown in Fig. 12.

Fig. 12. FL, SL and SFLV1 comparison when the number of Raspberry Pi devices (clients) varies from two to five.
decrease as it is highly related to the local data size. This corroborates with the statistical analysis result in a recent work [43], where the communication overhead of SL is shown significantly higher than that of FL per round for low model complexity and fewer clients.

For the used memory, as shown in Fig. 12c, the FL is always higher than that of SL and SFLV1 because the FL needs to train the entire model in the Raspberry Pi, while the rest two only need to train a small subnetwork. This also leads to the high power peak, as shown in Fig. 12d, and the high temperature of FL during training, as shown in Fig. 12e In the FL. Without cooling, the Raspberry Pi device’s temperature can be up to 83°C during the FL learning.

However, although FL has a high power peak, the energy is lesser than that of SL for the same number of rounds, as shown in Fig. 12d. This is because, in FL, each client trains the local model in parallel, and consequently, the total time (accumulated computation and communication) for running a given number of rounds is less than that of SL.

Notes: (i) The computational overheads, including used memory, peak power, and temperature of SL are always lower than that of FL. (ii) The training time and communication overhead of SL in our setting (small model, a limited number of clients) are worse than that of FL. (iii) SFLV1 shares the same overhead cost except for the training time, which is greatly reduced by the SFLV1 that is the main impetus of adopting in the IoT application.

To be precise, in each round, the communication overhead of the SL is dependent on the size of sent activation and received gradients, both are further determined by the output size of the cut layer and the size of the localized data. In contrast, in each round, the communication overhead of the FL relies on the model size, regardless of the size of the local data. Therefore, the SL has an advantage when the model size is large, the data held by each device is small, and the number of clients is larger. In other words, the communication overhead is constant as long as the cut layer output size and the local data size are fixed, even though the model size is increasing—SL can allocate the majority of layers to the server-side, thus offloading heavy computation to the server.

6 OPTIMIZATIONS

In this part, we perform two optimizations: the first is to generalize the SFLV1 and SFLV2, which both are inclusive of the presented generalized SFL (SFLG). The SFLG allows the server to flexibly manage IoT devices when they are large-scaled according to its memory resource, which is validated through performance comparison with SFLV1 and SFLV2. The second is to reduce the communication overhead of the SFLG when the communication bandwidth is bottlenecked, where the same optimization technique is applicable for SL.

6.1 SFL Generalization

Though it is assumed that the server is always with rich resources, it may still be undesirable to make a server-side sub-network copy corresponding to each device, especially when the number of devices is becoming large, e.g., tens of thousands. Therefore, the scalability of SFLV1 could still be a concern for large-scale devices. In addition, it is shown that the learning performance of SFLV1 is close to FL, where it converges slowly in most cases compared to SL.

To improve the scalability and learning performance, we have proposed a generalized SFLG (as detailed in Section 3) that fits the IoT scenario with flexible configurations. Below, we demonstrate its advantage over two specific SFL variants of SFLV1 and SFLV2 [16], which descriptions are referred to Section 2.3.

As a recall, the implementation of SFLG is as below:

1) the server makes a number of copies, $n$, of the server-side subnetwork, and each subnetwork copy is in charge of a group IoT devices.
2) within a group of IoT devices, the SFLV2 is applied; whereas among the server-side subnetwork copies, the SFLV1 is used.

For example, there are 100 clients in total. The server makes $n = 5$ server-side subnetwork copies to gain parallelization. This means these devices smashed data will be divided into 5 groups. In each group, one server-side subnetwork is in charge of 20 devices. Specifically, within a group of 20 IoT devices, one server-side subnetwork will sequentially make an update on each IoT device’s smashed data, taking the sequential training process of SL. Meanwhile, different 5 copies of the server-side network run in parallel from the server’s view, alike to the FL training process.

Results. Now we compare the learning performance of split fed learning, including SFLG, the results are depicted in Fig. 13 with IID data distribution, Fig. 14 with imbalanced data distribution, and Fig. 15 with non-IID data distribution.

For Fig. 13 with ECG dataset, we observe that the SFLV2 demonstrates the best learning performance as it converges the fastest, and the accuracy is also the highest given the same rounds used. In contrast, the SFLV1 is with the least performance. Interestingly, the SFLV2, in fact, approaches the SL, while the SFLV1 approaches the FL. SFLV1 avoids the spikes that existed in the SL, demonstrating a more stable learning curve and no turn-down compared with Fig. 3. Similar trends are also true for Figs. 14 and 15 under differing data distributions. In fact, when the number of the server-side network is set to be 1, the SFLG becomes SFLV2; when the number of the server-side network is equal to the number of IoT devices, the SFLG is SFLV1. This exactly corroborates our Preposition 1 in Section 3.

Notes: The learning performance of the proposed SFLG is bounded by SFLV2 as upper-bound and SFLV1 as lower-bound given any tested data distribution settings.
This has been validated in Figs. 14 and 15, where the SFLG(n) always lies between SFLV1 and SFLV2 detailed in the 100 clients case. More specifically, a small n, e.g., 10 in Fig. 14, makes SFLG to approach SFLV2, and a large n, e.g., 50 in Fig. 14, makes SFLG to approach SFLV1.

As we have shown that the SFLV1 shares the learning performance with the FL (Fig. 4), we have expected that the SFLV2 shares the learning performance with SL. While the latter is almost always the case, the SFLV2 is eventually with a relatively better performance with SL. Since the SFLV2 (i) eliminates the unstable spikes of the learning curve exhibited in the SL and (ii) avoids the turn-down learning curve of the SL. The potential reason lies in the fact that the device-side subnetworks now have an average operation, which enhances the feature representations sharing learned across clients.

The server plays the role of deciding the number of sub-networks. This determination is a trade-off among several factors including the resources of the server, the number of IoT devices, the expected training time, and the availability of edge servers. Though provisioning accurate quantitative formulation is uneasy, there are qualitative observations that can be followed. Firstly, the higher the resource of the server, the more sub-networks can run in parallel to expedite the training. Secondly, the larger the scaled IoT devices, the more sub-networks are preferable to enhance the parallelization. In any of the above two cases, the edge servers are always desirable to complement the main server while reducing the communication overhead latency between the edge server and devices charged by it. Take the second case as an example, when the IoT devices are large-scaled, e.g., millions, the |G| (number of server-side subnetwork copies, n) needs to be carefully chosen. Let M be the memory resource required for 1 subnetwork copy at the server-side, now if there are |G| ≥ 1 models, then there is |G| × M memory volume required to store the copies for the specific SFLV1, which still appears to be out of a common server’s memory capability—this volume could be up to hundreds GB when the |G| is hundreds of thousands given that one sub-network is as small as 1MB. Without SFLG that enables flexible choice of |G|, the intuitive setting of making a server-side subnetwork copy under SFLV1 is still prohibitive for the server, even considering its rich-resource when the IoT devices are large-scaled. With SFLG, an acceptable |G| can be chosen to ensure that large-scale IoT devices can be handled while still providing reasonable parallelization.

6.2 SL and SFLG Communication Overhead Reduction

Usually, in practice (e.g., IoT environment), communication is a bottleneck. So it is paramount to reduce the communication in the IoT settings for the SL. Notably, all the communication overhead optimization directly applies to approaches leveraging split architectures, including SFLG. We are interested in the following question.

Is there any efficient means of reducing SL communication overhead while retaining the accuracy?

The investigations below are affirmative by reducing the intermediate model output (cut layer) data size required to be communicated between the device and the server.

Given an input size of $A_{in}$, the output size after the pooling layer is expressed:

$$A_{out} = A_{in} - f + 1,$$  \hspace{1cm} (1)

with $f$ the kernel size and the $s$ the stride of the pooling layer.

In general, the communication overhead is linear to the ratio of:

$$\text{factor} = \frac{A_{out}}{A_{in}},$$  \hspace{1cm} (2)

with factor measures the communication overhead reduction. Smaller it, better or less communication overhead. Therefore, given the $A_{in}$, it is preferable to have a small $A_{out}$ by tuning the pooling layer parameter to facilitate communication.

The key is to reduce the output size of the cut layer on the device-side. Both the convolutional layer and pooling layers can affect the output size given input size by using different parameter settings, e.g., kernel size $f$ and stride $s$. We can flexibly select suitable parameter settings that reduce the cut layer’s output size while retaining the model accuracy. Below, we focus on the pooling layer for extensive experimental validations. We consider three settings:

- **Setting 1**: Rearrange or move the pooling layers of the server subnetwork to the device.
- **Setting 2**: Tuning the pooling parameter in the device subnetwork while fixing the server subnetwork.
- **Setting 3**: Fixing the device subnetwork while tuning server subnetwork to improve accuracy if setting 1 and 2 incur accuracy drop.

6.2.1 Setting 1

In this setting, we consider moving the pooling layer in the server subnetwork to the device-side. In particular, we move it to the cut-layer of the device-side, which is detailed in Table 2 (No. 2 compared with No. 1 as baseline). We can see $2 \times$ reduction in communication overhead. Note that, due to the position change of the pooling layer, the input size to the first dense layer (or fully connected layer) at the server subnetwork is also changed. As for the accuracy, we observe a slight decrease of about 1%, which tends to be acceptable considering the significant communication overhead decrease. If it does matter, this slight accuracy drop can be compensated via setting 3 detailed soon.

![Fig. 14. Testing accuracy SFLV1, SFLV2, and SFLG over rounds with imbalanced data setting on the ECG dataset.](image_url)

![Fig. 15. Testing accuracy of SFLV1, SFLV2, and SFLG over rounds with non-IID dataset setting on the SC dataset. There are 10 clients in total, the SFLG(#) means # server-side subnetwork replicas are used. Each server-side subnetwork is in charge of $\frac{1}{P}$ clients.](image_url)
TABLE 2
Tuning the Split Model Architecture to Reduce Communication Overhead

| No. | Device Network | Server Network | Comm. Overhead (Reduced by) | Factor | Accuracy |
|-----|----------------|----------------|-----------------------------|--------|----------|
| 1   | conv1d(16,4,6) + Fc(20,2) + conv1d(16,4,5) | conv1d(16,4,5) + Fc(20,2) + conv1d(16,4,5) | 39,770 MB | N/A | 94.50% |
| 2   | conv1d(16,4,6) + Fc(20,2) + conv1d(16,4,5) | conv1d(16,4,5) + Fc(20,2) + conv1d(16,4,5) | 9,395 MB (50.26%) | 50% (2) | 95.00% |
| 3   | conv1d(16,4,6) + Fc(20,2) + conv1d(16,4,5) | conv1d(16,4,5) + Fc(20,2) + conv1d(16,4,5) | 9,395 MB (50.26%) | 50% (2) | 97.10% |
| 4   | conv1d(16,4,6) + Fc(20,2) + conv1d(16,4,5) | conv1d(16,4,5) + Fc(20,2) + conv1d(16,4,5) | 9,395 MB (50.26%) | 50% (2) | 96.60% |
| 5   | conv1d(16,4,6) + Fc(20,2) + conv1d(16,4,5) | conv1d(16,4,5) + Fc(20,2) + conv1d(16,4,5) | 9,395 MB (50.26%) | 50% (2) | 94.90% |
| 6   | conv1d(16,4,6) + Fc(20,2) + conv1d(16,4,5) | conv1d(16,4,5) + Fc(20,2) + conv1d(16,4,5) | 9,395 MB (50.26%) | 50% (2) | 95.00% |
| 7   | conv1d(16,4,6) + Fc(20,2) + conv1d(16,4,5) | conv1d(16,4,5) + Fc(20,2) + conv1d(16,4,5) | 9,395 MB (50.26%) | 50% (2) | 95.20% |
| 8   | conv1d(16,4,6) + Fc(20,2) + conv1d(16,4,5) | conv1d(16,4,5) + Fc(20,2) + conv1d(16,4,5) | 9,395 MB (50.26%) | 50% (2) | 97.00% |

1. The leakyrelu activation is used after each conv1d layer, but not shown for concise purpose.
2. Device communication (Comm.) overhead is the data size sent to and received from the server. The number of epochs is 200.
3. conv1d(#input channel, #output channel, #kernel size, #padding). Pool (#kernel size, #stride). The output size of the second conv1d is 58 × 16, given the input size to the network is 1 × 130.

6.2.2 Setting 2

In this setting, we tune the pooling layer parameter as detailed in Table 2. As shown in No. 4, 5, and 6, the kernel size of the pooling is changed from 2 to 4, 6, and 8, respectively, while retaining the stride being unchanged. This gradually decreases the communication overhead to 48.28%, 46.55%, and 44.82%.

In No. 7, the stride is changed from 2 to 4 while the kernel size is retained to be 2. Such stride change results in significant communication overhead reduction—4× reduction in comparison with baseline in No. 1. Notably, the server subnetwork does require changing the input size of the first dense layer for the sake of size compatibility. As for the accuracy, it slightly drops. Again, this can be compensated by below setting 3.

6.2.3 Setting 3

This is to compensate for the slight accuracy drop resulted from setting 1 or/and 2 if the accuracy requirement is stringent. The insight here is to only fine-tune the model architecture in the server subnetwork without bringing any overhead to the device side. As detailed in No. 3 and 8, the number of filters is increased in the two conv1d layers—making the server subnetwork wider. As a consequence, the accuracy is improved, which is in fact slightly higher than the baseline in No. 1. In this context, the communication overhead reduction is reserved while the accuracy is retained without any additional overhead brought to the device.

The rationale behind setting 3 is as below. It has been proven that the model accuracy can be improved by generally increasing network width and depth [44]. In CNN, the width is related to the number of filters. Therefore, we can consider using a higher number of filters in the server-side network to compensate slight accuracy drop resulting from reducing the output size of the device-side subnetwork—the output size of the cut layer—to optimize the communication overhead. Though here we empirically configure the width setting, this configuration can be potentially automated via neural architecture search [45] in practice. In addition, one can also take increasing the depth of the server-side subnetwork into consideration. Notably, both the width and depth increase incur no computational overhead to the device side. Moreover, our proposal of reducing the size of the cut-layer is complementary with other distinct methods [46], which can be incorporated together to further reduce the SL communication overhead.

In Table 2, we have also experimentally measured the communication overhead besides the theory analysis in (2). While they agree well, the experimental reduction is slightly higher than the analysis in (2). The reason is that the subnetwork on the device-side needs to be sent to the server-side for updating per epoch, which results in additional small communication overhead that is not counted by the analysis in (2). Overall, by properly configuring the network resided in the device-side (e.g., pooling layers) and in the server-side (e.g., network width), we have experimentally validated that the communication overhead in SL can be substantially decreased without affecting the model accuracy and bringing a burden to the resource-restrict device. The training time and power consumption are also linearly related to the communication overhead given the same communication bandwidth. Reducing the communication overhead will also reduce the training time and power consumption.

Notes: Unlike the FL, which communication overhead is agnostic to the localized data—it is only dependent on the given (entire) model size, SL and SFLG are dependent on the size of the cut layer and the number of forward and backward propagation, where the later is related to the size of the localized data.

Therefore, it is worth considering i) means of reducing the size of the cut layer as we have validated and ii) further reducing the size of the localized data in future investigations. The latter can be realized by, e.g., principal component analysis, reducing the feature size to the data before feeding to the local subnetwork. This could have two benefits: reduction in communication overhead and reduction of the on-device computation over input with low feature dimensions.

The SFLG incorporates the FL tree-like architecture to mitigate the SL’s disadvantage of linear architecture to a large extent. The FL allows each local model to update in parallel. Note for SFLG, it can exploit the edge-computing resource such that an edge server manages a group of neighboring IoT devices (similar to SFLV2) to relax the communication bandwidth, while these edging servers run in parallel for training time expedition (similar to SFLV1). More generally, the server-side sub-networks are implemented in the edge-server. After step 3 as shown in Fig. 2 or at the end of each round, the edge-server first aggregates the server-side sub-networks and then sends them to the

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on September 22,2022 at 04:33:40 UTC from IEEE Xplore. Restrictions apply.
main-server for final aggregation. The device-side sub-networks can be processed in a similar manner.

In addition, it is expected that the 5G with the high data rate, more bandwidth will be widely used in the future—it is readily becoming available [47]. In this context, it is desirable to choose SFLG for large-scale IoT applications as it relaxes the on-device computation overhead.

7 Conclusion
This work is the first to provide an end-to-end comparison of FL, SL, and SFL in real-world IoT settings. We experimentally analyzed the learning performance in terms of model accuracy and convergence speed for each of them. We considered imbalanced and non-IID distributions for our experiments, which resembled real-world IoT scenarios. The FL and SL have their own advantages and disadvantages in terms of learning performance—potential rationales behind their performance, e.g., under non-IID data distribution, are provided.

Therefore, we proposed SFLG based on the advantage of both SL and FL so that a more reasonable distributed learning model can be deployed for IoT applications. We extensively evaluated the practicality of mounting the training of FL, SL, and SFL on resource-restricted computing platforms, i.e., Raspberry Pi.

We mainly dealt with pervasive sequential time-series data and provided useful comprehensive results—various implementation overhead—to the community. We corroborated the design impetus of SFLG to provide (i) lower computational overhead than FL and (ii) shorter training time than SL. We have further validated the flexibility of the proposed SFLG in comparison with two specific SFL variants. By considering the possible communication bottleneck in practice (e.g., IoT settings), we proposed pragmatic approaches to reduce the communication overhead of SL (also applicable for SFLG) and empirically validated the efficiency of those optimizations.

References
[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning.” Nature, vol. 521, no. 7553, pp. 436–444, 2015.
[2] M. Bakator and D. Radosav, “Deep learning and medical diagnosis: A review of literature,” Multimodal Technol. Interact., vol. 2, no. 3, 2018, Art. no. 47.
[3] T. A. Tang, L. Mhamdi, D. McLernon, S. A. Raza Zaidi, and M. Ghogho, “Deep learning approach for network intrusion detection in software defined networking,” in Proc. Int. Conf. Wireless Netw. Mobile Commun., 2016, pp. 258–263.
[4] L. Huang, A. L. Shea, H. Qian, A. Masurkar, H. Deng, and D. Liu, “Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records,” J. BioMed. Inform., vol. 99, 2019, Art. no. 103291.
[5] InvenSense Inc, “InvenSense announces world’s smallest, lowest power 125 microphone for IoT and wearable markets,” 2016, Accessed: Dec. 21, 2021. [Online]. Available: https://www.invensense.com/news-media/invensense-announces-worlds-smallest-lowest-power-125-microphone-for-iot-and-wearable-markets/.
[6] Europe, “General data protection regulation,” 2016, Accessed: Feb. 05, 2021. [Online]. Available: https://www.europarl.europa.eu/has/lawmaking/us/documents/adr/2016/0048_IN/adr-0048_EN.pdf.
[7] United States, “California privacy rights act,” 2020, Accessed: Feb. 05, 2021. [Online]. Available: https://www.cpr.org/.
[8] The Washington post, “Amazon alexa user receives 1,700 audio recordings of a stranger through human error,” 2018, Accessed: Dec. 21, 2021. [Online]. Available: https://www.washingtonpost.com/technology/2018/12/20/amazon-alexa-user-receives-audio-recordings-stranger-through-human-error/.
[9] S. Shastry, M. Wasserman, and V. Chidambaram, “The seven sins of personal-data processing systems under GDPR,” in Proc. 11th USENIX Conf. Hot Topics Cloud Comput., 2019, Art. no. 1.
[10] ZDNet, “Major biometrics data leak impacts UK metropolitan police, banks, enterprise companies,” 2019, Accessed: Dec. 21, 2021. [Online]. Available: https://www.zdnet.com/article/major-biometrics-data-leak-impacts-police-banks-enterprise-companies/.
[11] H. Brendan McMahan, “A survey of algorithms and analysis for adaptive online learning,” J. Mach. Learn. Res., vol. 18, pp. 901–9050, 2017.

Authorized license use limited to: Sungkyunkwan University. Downloaded on September 22, 2022 at 04:33:40 UTC from IEEE Xplore. Restrictions apply.
Alsharif Abuaddba received the PhD degree in computer security from RMIT University, Australia, 2017. He is currently a research scientist with CSIRO’s Data61 and Cybersecurity CRC fellow. He also has several years of experience working as a senior research engineer with Californian-based technology companies. He has contributions to a few U.S. IP filled Patents in cybersecurity. His specialist and interests include AI & cybersecurity, IoT-Cloud Security, System security, and watermarking.

Zhi Zhang received the PhD degree in computer science from the University of New South Wales. He is currently a postdoctoral fellow with CSIRO Data61. His research interests include system security, rowhammer, and adversarial artificial intelligence.

Seyit Camtepe received the PhD degree in computer science from Rensselaer Polytechnic Institute, in 2007. He is currently a principal research scientist with CSIRO Data61. He worked as a senior researcher with the Technische Universität Berlin (2007-13), and worked as a lecturer with the Queensland University of Technology (2013-17). His research interests include cyber security and ML, applied and malicious cryptography, wireless and mobile security.

Hyounghshick Kim received the BS degree from the Department of Information Engineering, Sungkyunkwan University, the MS degree from the Department of Computer Science, KAIST, and the PhD degree from the Computer Laboratory, University of Cambridge, in 1999, 2001, and 2012, respectively. He is an associated professor with the Department of Computer Science and Engineering, Sungkyunkwan University. He is also a visiting scientist with Data61, CSIRO, in 2019. After completing his PhD, he worked as a postdoctoral fellow with the Department of Electrical and Computer Engineering, University of British Columbia. He previously worked for Samsung Electronics as a senior engineer from 2004 to 2008. His current research interest includes usable security and security engineering.

Surya Nepal received the bachelor’s degree from the National Institute of Technology, Surat, India, the master’s degree from the Asian Institute of Technology, Bangkok, Thailand, and the PhD degree from RMIT University, Australia. He joined CSIRO in 2000. He is currently a principal research scientist with Data61, CSIRO. He is the leader of the Distributed System Security Group, Data61, CSIRO. His main research interests include the development and implementation of technologies in the area of distributed systems, including Web services, cloud computing, IoT, and big data, with a specific focus on security, privacy, and trust. He is currently one of the associate editors of the IEEE Transactions on Dependable and Secure Computing and IEEE Transactions on Services Computing.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.

Yansong Gao received the MSc degree from the University of Electronic Science and Technology of China, in 2013, and the PhD degree from the University of Adelaide, Australia, in 2017. His current research interests include AI security and privacy, hardware security, and system security.

Minki Kim received the BS degree from the College of Computing, Sungkyunkwan University, South Korea, in 2021. He is currently a software engineer of Samsung Electronics. His current research interests include deep learning, AI privacy, and computer network.

Chandra Thapa received the PhD degree from the University of Newcastle, Australia, in 2018. He is currently a postdoctoral fellow within CSIRO Data61. His research interests include the field of privacy-preserving computation, distributed systems security, and network information theory. His current works include data security, privacy, and the application of privacy-preserving approaches to machine learning/artificial intelligence in the health domain.

[34] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous networks,” in Proc. Mach. Learn. Syst., 2020, vol. 2, pp. 429–450.
[35] T. Li, A. Kumar 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.
[36] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, “On the convergence of fedavg on non-IID data,” in Proc. Int. Conf. Lear. Representations, 2020.
[37] R. M. French, “Catastrophic forgetting in connectionist networks,” Trends Cogn. Sci., vol. 3, no. 4, pp. 128–135, 1999.
[38] M. J. Sheller, “Advances and open problems in federated learning,” IEEE Transactions on Services Computing, vol. 20, no. 1, pp. 1997–2017, 2020.
[39] K. Chang, et al., “Distributed deep learning networks among institutions for medical imaging,” J. Amer. Med. Inform. Assoc., vol. 25, no. 8, pp. 945–954, 2018.
[40] H. Madaan, M. Gawli, V. Kulkarni, and A. Pant, “Vulnerability due to training order in split learning,” 2021, arXiv:2103.14291.
[41] P. Kairouz et al., “Advances and open problems in federated learning,” 2019, arXiv:1912.04977.
[42] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, “DiIoT: A federated self-learning anomaly detection system for IoT,” in Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst., 2019, pp. 756–767.
[43] A. Singh, P. Vepakomma, O. Gupta, and R. Raskar, “Detailed comparison of communication efficiency of split learning and federated learning,” 2019. [Online]. Available: https://arxiv.org/abs/1909.09145
[44] M. Tan and Q. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 6103–6114.
[45] E. Elsken, J. HendrikMetzen, and F. Hutter, “Neural architecture search: A survey,” J. Mach. Learn. Res., vol. 20, no. 1, pp. 1997–2017, 2019.
[46] X. Chen, J. Li, and C. Chakrabarti, “Communication and computation reduction for split learning using asynchronous training,” 2021, arXiv:2107.09786.
[47] L. Chetturi and R. Bera, “A comprehensive survey on internet of things (IoT) toward 5G wireless systems,” IEEE Internet Things J., vol. 7, no. 1, pp. 16–32, Jan. 2020.