Model Pruning Enables Efficient Federated Learning on Edge Devices

Yuang Jiang, Shiqiang Wang, *Member, IEEE*, Víctor Valls, Bong Jun Ko, Wei-Han Lee, Kin K. Leung, *Fellow, IEEE* and Leandros Tassiulas, *Fellow, IEEE*

***Abstract*— Federated learning (FL) allows model training from local data collected by edge/mobile devices while preserving data privacy, which has wide applicability to image and vision applications. A challenge is that client devices in FL usually have much more limited computation and communication resources compared to servers in a data center. To overcome this challenge, we propose *PruneFL*—a novel FL approach with adaptive and distributed parameter pruning, which adapts the model size during FL to reduce both communication and computation overhead and minimize the overall training time, while maintaining a similar accuracy as the original model. PruneFL includes initial pruning at a selected client and further pruning as part of the FL process. The model size is adapted during this process, which includes maximizing the approximate empirical risk reduction divided by the time of one FL round. Our experiments with various datasets on edge devices (e.g., Raspberry Pi) show that: 1) we significantly reduce the training time compared to conventional FL and various other pruning-based methods and 2) the pruned model with automatically determined size converges to an accuracy that is very similar to the original model, and it is also a lottery ticket of the original model.**

***Index Terms*— Efficient training, federated learning (FL), model pruning.**

I. Introduction

THE past decade has seen a rapid development of machine learning algorithms and applications, particularly in the area of deep neural networks (DNNs) [1]. However, a huge volume of training data is usually required to train accurate models for complex tasks, such as image classification and computer vision. Due to limits in data privacy regulations and communication bandwidth, it is usually infeasible to transmit and store all training data at a central location. To address this problem, *federated learning* (FL) has emerged as a promising approach to distributed model training from decentralized data [2]–[6]. In a typical FL system, data are collected by client devices (e.g., cameras) at the network edge; the training process includes local model updates using each client's own data and the fusion of all clients' models typically through a server. In this way, the raw data remain local to clients.

Client devices in FL are usually much more resource-constrained than server machines in a data center in terms of computation power, communication bandwidth, memory and storage size, and so on. Training DNNs that can include millions of parameters (weights) on such resource-limited edge devices can take prohibitively long and consume a large amount of energy. Therefore, a natural question is: *how can we perform FL efficiently so that a model is trained within a reasonable amount of time and energy?*

Some progress has been made toward this direction recently using model/gradient compression techniques, where, instead of training the original model with a full parameter vector, either a small model is extracted from the original model for training or a compressed parameter vector (or its gradient) is transmitted in the fusion stage [7]–[10]. However, the former approach may reduce the accuracy of the final model in undesirable ways, whereas the latter approach only reduces the communication overhead and does not generate a small model for efficient computation. Furthermore, how to adapt the compressed model size for the most efficient training remains a largely unexplored area, which is a challenging problem due to unpredictable training dynamics and the need of obtaining a good solution in a short time with minimal overhead.

To overcome these problems, we propose a new FL paradigm called *PruneFL*, which includes *adaptive and distributed parameter pruning* as part of the FL procedure. We make the following key contributions.

A. Distributed Pruning

PruneFL includes initial pruning at a selected client followed by further distributed pruning that is intertwined with the standard FL procedure. Our experimental results show that this method outperforms alternative approaches that either prune at a single client only or directly involve multiple FL clients for pruning, especially when the clients have heterogeneous data statistics and computational power.

B. Adaptive Pruning

PruneFL continuously "tracks" a model that is small enough for efficient transmission and computation with a low memory

Manuscript received June 1, 2021; revised October 27, 2021 and February 2, 2022; accepted April 1, 2022. This work was supported in part by the U.S. Office of Naval Research under Grant N00014-19-1-2566, in part by the U.S. National Science Foundation AI Institute Athena under Grant CNS-2112562, and in part by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement W911NF-16-3-0001. The work of Víctor Valls was supported by the European Union's Horizon 2020 Research and Innovation Program through the Marie Skłodowska-Curie Grant under Agreement 795244. *(Corresponding author: Yuang Jiang.)*

Yuang Jiang, Víctor Valls, and Leandros Tassiulas are with the Department of Electrical Engineering, Yale University, New Haven, CT USA (e-mail: yuang.jiang@yale.edu).

Shiqiang Wang and Wei-Han Lee are with the IBM T. J. Watson Research Center, Yorktown Heights, NY USA (e-mail: wangshiq@us.ibm.com).

Bong Jun Ko was with the Stanford Institute for Human-Centered Artificial Intelligence (HAI), Stanford, CA USA. He is now with the Visual Display Division, Samsung Electronics, Suwon-Si, Gyeonggi-Do 443-743, South Korea.

Kin K. Leung is with the Department of Electrical and Electronic Engineering, Imperial College London, London, U.K.

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2022.3166101.

Digital Object Identifier 10.1109/TNNLS.2022.3166101

footprint while maintaining useful connections and their parameters so that the model converges to a similar accuracy as the original model. The importance of model parameters evolves during training, so our method continuously updates which parameters to keep and the corresponding model size. The update follows an objective of minimizing the time of reaching intermediate and final loss values. Each FL round operates on a small pruned model, which is efficient. A small model is also obtained at any time during and after the FL process for efficient inference on edge devices, which is a lottery ticket [11] of the original model as we show experimentally.

C. Implementation

We implement FL with model pruning on real edge devices, where we extend a deep learning framework to support efficient sparse matrix computation. Our code is available at https://github.com/jiangyuang/PruneFL.

II. Related Work

A. Neural Network Pruning

To reduce the complexity of neural network models, different ways of parameter pruning were proposed in the literature. Early work considered approximation using second-order Taylor expansion [12]. However, the computation of the Hessian matrix has high complexity, which is infeasible for modern DNNs. In recent years, magnitude-based pruning has become popular [13], where parameters with small enough magnitudes are removed from the network. A finding that suggests a network that is pruned by magnitude consists of an optimal substructure of the original network, known as the "lottery ticket hypothesis," was presented in [11] and [14]. It shows that directly training the pruned network can reach a similar accuracy as pruning a pretrained original network.

In addition to the above approaches that train until convergence before the next pruning step, there are iterative pruning methods where the model is pruned after every few steps of training [15], [16]. There are also one-shot pruning approaches including SynFlow [17] that prunes the model at model initialization (before training) and SNIP [18] that prunes the model using the first training round's gradient information. A dynamic pruning approach that allows the network to grow and shrink during training was proposed in [19]. Besides these unstructured pruning methods, structured pruning was also studied [20], which, however, often requires specific network architectures and does not conform to the lottery ticket hypothesis. The lottery ticket is useful for retraining a pruned model on a different yet similar dataset [14]. The use of pruning for efficient model training was discussed in [21], where the optimal choice of pruning rate (or final model size) remained unstudied.

These existing pruning techniques consider the centralized setting with full access to training data, which is fundamentally different from our PruneFL that works with decentralized datasets at local clients. Furthermore, the automatic adaptation of model size has not been studied before.

B. Efficient Federated Learning

The first FL method is known as federated averaging (FedAvg) [2], where each "round" of training includes multiple local gradient computation steps on each client's local data, followed by a parameter averaging step through a server. This method can be shown to converge in various settings including when the data at different clients are nonidentically distributed (non-IID) [22]–[24].

To improve the communication efficiency of FL, methods for optimizing the communication frequency were studied [25]–[27]. An approach of parameter averaging using structured, sketched, and quantized updates was introduced in [7], which belongs to the broader area of gradient compression/sparsification [28]–[33]. These techniques usually consider a fixed degree of sparsity or compression that needs to be configured as a hyperparameter. An online learning approach that determines a near-optimal gradient sparsity was proposed in [9], which includes exploration steps that may slow down the training initially. This body of work does not address computation efficiency.

To reduce both communication and computation costs, efficient FL techniques using lossy compression and dropout were developed [8], [10], where the final model still has the original size, hence providing no benefit for efficient inference after the model is trained. Moreover, because the main goal of pruning is to remove less important weights from the model, it is orthogonal to other acceleration methods, such as quantization [34] and low-rank decomposition [35], and pruning can be applied together with these other methods. In addition, since our approach considers acceleration for both training and inference, methods that accelerate inference only, e.g., knowledge distillation [36], runtime neural pruning [37], and DNN partitioning and offloading [38], do not serve our purpose. There are other distributed training methods, such as split learning [39], which are beyond our scope since we focus on FL in this article.

Furthermore, most existing studies on FL are based on simulation. Only a few recent articles considered implementation on real embedded devices [10], [26], but they do not include parameter pruning.

C. Novelty of Our Work

The uniqueness of PruneFL is that we jointly address communication and computation efficiency for both training and inference phases by extending FedAvg with minimal extra overhead. Our two-stage distributed pruning method is designed to address both data (statistical) and device (system) heterogeneity, including non-IID data distribution. Our adaptive pruning method is uniquely based on gradient information, which does not require sharing clients' local data, so that existing privacy preservation and secure aggregation [40] methods for FL can be directly applied to the gradient. Thus, our approach does not introduce extra privacy concerns.

D. Roadmap

The remainder of this article is organized as follows. Section III provides preliminaries of FL and model pruning. Section IV presents the proposed PruneFL approach and its analysis. Implementation challenges are discussed in Section V. Section VI presents the experimental setup and results. Section VII draws conclusion.

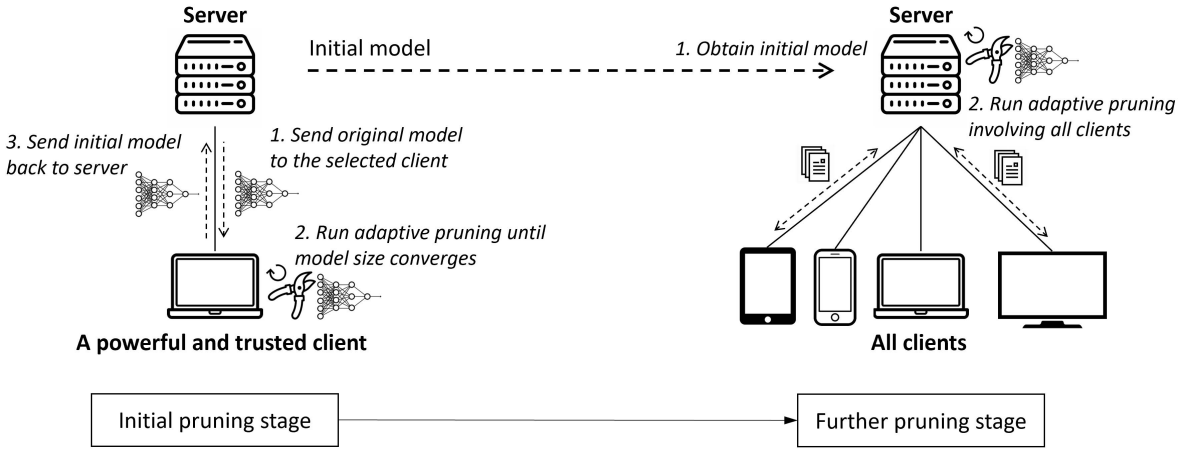


Fig. 1. Illustration and flowchart of PruneFL.

III. Preliminaries

A. Federated Learning

We consider an FL system with N clients. Each client $n \in [N] := \{1, 2, \ldots, N\}$ has a local empirical risk $F_n(\mathbf{w}) := (1/D_n) \sum_{i \in \mathcal{D}_n} f_i(\mathbf{w})$ defined on its local dataset $\mathcal{D}_n$ ($D_n := |\mathcal{D}_n|$) for model parameter vector $\mathbf{w}$, where $f_i(\mathbf{w})$ is the loss function (e.g., cross-entropy and mean square error) that captures the difference between the model output and the desired output of data sample i. The system tries to find a parameter $\mathbf{w}$ that minimizes the global empirical risk

$$\min_{\mathbf{w}} \; F(\mathbf{w}) := \sum_{n \in [N]} p_n F_n(\mathbf{w}) \tag{1}$$

where $p_n > 0$ are weights such that $\sum_{n \in [N]} p_n = 1$. For example, if $\mathcal{D}_n \cap \mathcal{D}_{n'} = \emptyset$ for $n \neq n'$ and $p_n = D_n/D$ with $\mathcal{D} := \bigcup_n \mathcal{D}_n$ and $D := |\mathcal{D}|$, we have $F(\mathbf{w}) = (1/D) \sum_{i \in \mathcal{D}} f_i(\mathbf{w})$. Other ways of configuring p_n may also be used to account for fairness and other objectives [41].

In FL, each client n has a local parameter $\mathbf{w}_n(k)$ in iteration k. The aggregation of these local parameters is defined as $\mathbf{w}(k) := \sum_{n \in [N]} p_n \mathbf{w}_n(k)$. The FL procedure usually involves multiple updates of $\mathbf{w}_n(k)$ using stochastic gradient descent (SGD) on the local empirical risk $F_n(\mathbf{w}_n(k))$ computed by every client n, followed by a parameter fusion step that involves the server collecting clients' local parameters $\{\mathbf{w}_n(k) : \forall n \in [N]\}$ and computing the aggregated parameter $\mathbf{w}(k)$. After parameter fusion, the local parameters $\{\mathbf{w}_n(k) : \forall n \in [N]\}$ are all set to be equal to the aggregated parameter $\mathbf{w}(k)$.

In the following, we call this procedure of multiple local SGD iterations followed by a fusion step a *round*. We use I to denote the number of local SGD iterations in each round. The main notations in this article are listed in Table I.

It is possible that each round only involves a subset of clients, to avoid the excessive delay caused by waiting for all the clients [42]. It has been shown that FedAvg converges even with random client participation although the convergence rate is related to the degree of such randomness [23].

B. Model Pruning

In the iterative training and pruning approaches for the centralized machine learning setting, the model is first trained using SGD for a given number of iterations [13], [15], [16]. Then, a certain percentage (referred to as the pruning rate) of weights that have the smallest absolute values within each layer is removed (set to zero). This training and pruning process is repeated until the desired model size is reached. The benefit of this approach is that the training and pruning occur at the same time so that a trained model with a desired (small) size can be obtained in the end. However, existing pruning techniques require the availability of training data at a central location, which is not applicable to FL.

TABLE I
Main Notations

Notation	Definition
$\odot$	element-wise product of two vectors
n, N	client index, total number of clients
k, K	iteration index, total number of iterations
I	number of local iterations
p_n	weight for client n ($p_n > 0$, $\forall n$, and $\sum_n p_n = 1$)
$\mathbf{m}(k)$	weight mask in iteration k (universal for all clients)
$\mathbf{w}_n(k)$	client n's parameter in iteration k
$\mathbf{w}(k)$	$\mathbf{w}(k) := \sum_{n=1}^{N} p_n \mathbf{w}_n(k)$
$\mathbf{w}'_n(k)$, $\mathbf{w}'(k)$	$\mathbf{w}'_n(k) = \mathbf{w}_n(k) \odot \mathbf{m}(k)$, $\mathbf{w}'(k) = \mathbf{w}(k) \odot \mathbf{m}(k)$
$\mathbf{g}_n(\mathbf{w})$	client n's stochastic gradient with parameter $\mathbf{w}$
$\nabla F_n(\mathbf{w})$	client n's expected gradient with parameter $\mathbf{w}$
$\nabla F(\mathbf{w})$	$\nabla F(\mathbf{w}) = \sum_{n=1}^{N} p_n F_n(\mathbf{w})$

IV. PruneFL

Our proposed PruneFL approach includes two stages: initial pruning at a selected client and further pruning involving both the server and clients during the FL process. The initial pruning can be done with biased data at a single client that has a relatively high computational capability, and the further pruning stage will "remove" the bias and refine the model. We use *adaptive pruning* in both stages. The illustration of the overall procedure is presented in Fig. 1. In the following, we introduce the two pruning stages (see Section IV-A) and adaptive pruning (see Section IV-B).

Algorithm 1 Adaptive Pruning

for $k = 0, \ldots, K-1$ **do**
  Initialize the set of importance measure on each client: $\mathcal{Z}_n \leftarrow \emptyset, \forall n$;
  for *each client n, in parallel* **do**
    Compute stochastic gradient $\mathbf{g}_n(\mathbf{w}'_n(k)) := \mathbf{g}_n(\mathbf{w}_n(k) \odot \mathbf{m}(k))$;
    Update local parameters: $\mathbf{w}_n(k+1) \leftarrow \mathbf{w}'_n(k) - \eta \mathbf{g}_n(\mathbf{w}'_n(k)) \odot \mathbf{m}(k)$;
    Add importance measure $\mathbf{z}_n$ to $\mathcal{Z}_n$: $\mathbf{z}_n := \mathbf{g}_n(\mathbf{w}'_n(k)) \odot \mathbf{g}_n(\mathbf{w}'_n(k))$; $\mathcal{Z}_n \leftarrow \mathcal{Z}_n \cup \mathbf{z}_n$;
  if $I \mid k+1$ **then**
    Each client n sends $\mathbf{w}_n(k+1)$ to the server;
    Server aggregates the parameters from each client: $\mathbf{w}(k+1) \leftarrow \sum_{n=1}^{N} p_n \mathbf{w}_n(k+1)$;
    if $k+1$ *is reconfiguration iteration* **then**
      Each client sends the averaged importance measure $\overline{\mathbf{z}}_n := (\sum_{\mathbf{z}_n \in \mathcal{Z}_n} \mathbf{z}_n)/|\mathcal{Z}_n|$ to the server;
      Server aggregates the received importance measure: $\mathbf{z} \leftarrow \sum_{n=1}^{N} p_n \overline{\mathbf{z}}_n$;
      Reconfigure using Algorithm 2: $\mathbf{w}'(k+1), \mathbf{m}(k+1) \leftarrow \text{reconfigure}(\mathbf{w}(k+1), \mathbf{z})$;
      Reset: $\mathcal{Z}_n \leftarrow \emptyset, \forall n$;
    else
      No reconfiguration: $\mathbf{w}'(k+1) \leftarrow \mathbf{w}(k+1)$;
    Server sends new parameters to each client: $\mathbf{w}'_n(k+1) \leftarrow \mathbf{w}'(k+1), \forall n$;

A. Two-Stage Distributed Pruning

1) Initial Pruning at a Selected Client: Before FL starts, the system selects a single client to prune the model using its local data. This is important for two reasons. First, it allows us to start the FL process with a small model, which can significantly reduce the computation and communication time of each FL round. Second, when clients have heterogeneous computational capabilities, the selected client for initial pruning can be one that is powerful and trusted so that the time required for initial pruning is short. We apply the adaptive pruning procedure that we describe in Section IV-B, where we adjust the original model iteratively while training the model on the selected client's local dataset, until the model size remains almost unchanged.

2) Further Pruning During FL Process: The model produced by initial pruning may not be optimal because it is obtained based on data at a single client. However, it is a good starting point for the FL process involving all clients. During FL, we perform further adaptive pruning together with the standard FedAvg procedure, where the model can either grow or shrink depending on which way it makes the training most efficient. In this stage, data from all participating clients are involved.

B. Adaptive Pruning

For our adaptive pruning method, the notion of pruning broadly includes both removing and adding back parameters. Hence, we also refer to such pruning operations as *reconfiguration*. We reconfigure the model at a given interval of multiple iterations. For initial pruning, the reconfiguration interval can be any number of local iterations at the selected client. For further pruning, reconfiguration is done at the server after receiving parameter updates from clients (i.e., at the boundary between two rounds), and the reconfiguration interval, in this case, is always an integer multiple of the number of iterations (i.e., I) in each round.

1) Definitions: Let k denote the iteration index and $\mathbf{g}_n(\mathbf{w}(k))$ denote the stochastic gradient of $F_n(\mathbf{w}(k))$ evaluated at $\mathbf{w}(k)$ and computed on the full parameter space on client n. Also, let $\mathbf{m}_{\mathbf{w}}(k)$ denote a mask vector that is zero if the corresponding component in $\mathbf{w}(k)$ is pruned and one if not pruned, and $\odot$ denote the elementwise product.

In each reconfiguration step, adaptive pruning finds an optimal set of remaining (i.e., not pruned) model parameters. Then, parameters are pruned or added back accordingly, and the resulting model and mask are used for training until the next reconfiguration step. This procedure is illustrated in Algorithm 1, where $a \mid b$ denotes that a divides b, i.e., b is an integer multiple of a. More details are given in Appendix C. *All appendices are included as part of the online supplementary material.*

Our goal is to find the subnetwork that learns the "fastest." We do so by estimating the empirical risk reduction divided by the time required for completing an FL round, for any given subset of parameters chosen to be pruned. Note that, at the beginning of each round (after averaging the clients' parameters), all clients start with the same parameter vector $\mathbf{w}$, i.e., $\mathbf{w}_n(k) = \mathbf{w}(k)$ for all n if a new round starts at iteration k (see Section III). For approximation purpose, we first consider the change of empirical risk after *one SGD iteration* starting with a *common* parameter $\mathbf{w}(k)$ for both initial and further pruning stages. The full FL procedure will be considered later in Theorem 2.

When the model is reconfigured at the end of iteration k, parameter update in the next iteration will be done on the reconfigured parameter $\mathbf{w}'(k)$, so we have an SGD update step that follows:

$$\begin{aligned}\mathbf{w}(k+1) &= \mathbf{w}'(k) - \eta \sum_{n=1}^{N} p_n \mathbf{g}_n\big(\mathbf{w}'(k)\big) \odot \mathbf{m}_{\mathbf{w}'}(k) \\ &= \mathbf{w}'(k) - \eta \mathbf{g}\big(\mathbf{w}'(k)\big) \odot \mathbf{m}(k) \end{aligned} \tag{2}$$

where η is the learning rate. For simplicity, we define $\mathbf{g}(\mathbf{w}'(k)) := \sum_{n=1}^{N} p_n \mathbf{g}_n(\mathbf{w}'(k))$, and we omit the subscript $\mathbf{w}'$ of $\mathbf{m}$ in the following when it is clear from the context.

Let $\mathcal{M}$ denote the index set of components that are not pruned, which corresponds to the indices of all nonzero values of the mask $\mathbf{m}(k)$.

2) Empirical Risk Reduction: To analyze the empirical risk reduction, we use a first-order approximation, which is a common practice in the literature [18], [43], [44]. We have

$$\begin{aligned} &F(\mathbf{w}(k+1)) \\ &\quad\approx F\big(\mathbf{w}'(k)\big) + \langle \nabla F\big(\mathbf{w}'(k)\big), \mathbf{w}(k+1) - \mathbf{w}'(k) \rangle && (3) \\ &\quad= F\big(\mathbf{w}'(k)\big) - \eta \langle \nabla F\big(\mathbf{w}'(k)\big), \mathbf{g}\big(\mathbf{w}'(k)\big) \odot \mathbf{m}(k) \rangle && (4) \\ &\quad\approx F(\mathbf{w}(k)) - \eta \big\| \mathbf{g}\big(\mathbf{w}'(k)\big) \odot \mathbf{m}(k) \big\|^2 && (5) \end{aligned}$$

where $\langle \cdot, \cdot \rangle$ is the inner product, (3) is from Taylor expansion, (4) is because of (2), and (5) is obtained by using the stochastic gradient to approximate the actual gradient, i.e., $\mathbf{g}(\mathbf{w}'(k)) \approx \nabla F(\mathbf{w}'(k))$. Then, the approximate decrease of empirical risk after the SGD step (2) is

$$\begin{aligned} F\big(\mathbf{w}'(k)\big) - F(\mathbf{w}(k+1)) &\approx \eta \big\| \mathbf{g}\big(\mathbf{w}'(k)\big) \odot \mathbf{m}(k) \big\|^2 \\ &\propto \big\| \mathbf{g}\big(\mathbf{w}'(k)\big) \odot \mathbf{m}(k) \big\|^2 \\ &= \sum_{j \in \mathcal{M}} g_j^2 =: \Delta(\mathcal{M}) \end{aligned} \tag{6}$$

where g_j is the jth component of $\mathbf{g}(\mathbf{w}'(k))$, and we define the set function $\Delta(\mathcal{M})$ in the last line. The learning rate η is omitted since it is independent to the relative importance between parameter components, no matter if it is constant or varying. We use $\Delta(\mathcal{M})$ as the *approximate risk reduction*, where we ignore the proportionality coefficient because our optimization problem is independent of the coefficient. As $\Delta(\mathcal{M})$ is defined as the sum of g_j^2 in (6), we use g_j^2 as the *importance measure* for the jth component of the parameter vector.

Remark: During the further pruning stage, the stochastic gradient $\mathbf{g}(\mathbf{w}'(k))$ is the aggregated stochastic gradient from clients in FL. Since clients cannot compute $\mathbf{g}(\mathbf{w}'(k))$ before receiving $\mathbf{w}'(k)$ from the server, they compute $\mathbf{g}(\mathbf{w}(k))$, and we use $\mathbf{g}(\mathbf{w}'(k)) \approx \mathbf{g}(\mathbf{w}(k))$, both of which are denoted by $\mathbf{g}(\mathbf{w}'(k))$ with components $\{g_j\}$ in the following. The additional overhead for clients to compute and transmit gradients on the full parameter space in a reconfiguration is small because pruning is done once in many FL rounds (the interval between two reconfigurations is 50 rounds in our experiments). Further details are given in Appendix C.

3) Time of One FL Round: We define the (approximate) time of one FL round when the model has remaining parameters $\mathcal{M}$ as a set function $T(\mathcal{M}) := c + \sum_{j \in \mathcal{M}} t_j$, where $c \geq 0$ is a fixed constant and $t_j > 0$ is the time corresponding to the jth parameter component. Note that this is a linear function, which is sufficient according to our empirical observations (see Appendix D1). In particular, the quantity t_j has a value that can be dependent on the neural network layer, and c captures a constant system overhead. From our experiments, we observed that t_j remains the same for all j's that belong to the same neural network layer. Therefore, we can estimate the quantities $\{t_j\}$ and c by measuring the time of one FL round for a small subset of different model sizes, before the overall pruning and FL procedure starts. An extension to the general case with nonlinear $T(\mathcal{M})$ is also discussed in Appendix A.

4) Optimization of Reconfiguration: We would like to find the set of remaining parameters $\mathcal{M}$ that maximizes the empirical risk reduction per unit training time. However, $\Delta(\mathcal{M})$ only captures the risk reduction in the *next* SGD step when starting from the reconfigured parameter vector $\mathbf{w}'(k)$, as defined in (6). It does not capture the change in empirical risk when using $\mathbf{w}'(k)$ instead of the original parameter vector $\mathbf{w}(k)$ before reconfiguration. In other words, in addition to maximizing $\Gamma(\mathcal{M}) := (\Delta(\mathcal{M})/T(\mathcal{M}))$, we also need to ensure that $F(\mathbf{w}'(k)) \approx F(\mathbf{w}(k))$.

To ensure that $F(\mathbf{w}'(k)) \approx F(\mathbf{w}(k))$ after reconfiguration, we define an index set $\overline{\mathcal{P}}$ to denote the parameters that *are not allowed to* be pruned. Usually, $\overline{\mathcal{P}}$ includes parameters whose magnitudes are larger than a certain threshold because pruning them can cause $F(\mathbf{w}'(k))$ to become much larger than $F(\mathbf{w}(k))$. Among the remaining parameters that *can* be pruned (or added back if they are already pruned before), denoted by $\mathcal{P}$, we find which of them to prune to maximize $\Gamma(\mathcal{M})$. This yields the following optimization problem:

$$\max_{\mathcal{A} \subseteq \mathcal{P}} \ \Gamma\big(\mathcal{A} \cup \overline{\mathcal{P}}\big) \tag{7}$$

where $\mathcal{A}$ is the set of parameters in $\mathcal{P}$ that remain (i.e., are not pruned). The final set of remaining parameters is then $\mathcal{M} = \mathcal{A} \cup \overline{\mathcal{P}}$. Note that $\mathcal{P} \cup \overline{\mathcal{P}}$ is the set of all parameters in the original model.

Algorithm 2 Solving (7)

Input : importance measure g_j^2 and time coefficient t_j, for each parameter index j
Output: the optimal subset of parameters $\mathcal{A}$

$\mathcal{A} \leftarrow \emptyset$;
$\mathcal{S} \leftarrow \arg\operatorname{sort}_{j \in \mathcal{P}} \frac{g_j^2}{t_j}$; // ordered set
for $j \in \mathcal{S}$ **do**
  if $\frac{g_j^2}{t_j} \geq \Gamma(\mathcal{A} \cup \overline{\mathcal{P}})$ **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{j\}$;
  else
    break;
return $\mathcal{A}$; // final result

The algorithm for solving (7) is given in Algorithm 2, where sorting is in nonincreasing order and $\mathcal{S}$ is an ordered set that includes the sorted indices. In essence, this algorithm sorts the ratios of components in the sums of $\Delta(\mathcal{M})$ and $T(\mathcal{M})$. When the individual ratio g_j^2/t_j is larger than the current overall ratio Γ, then adding j to $\mathcal{A}$ increases Γ. The bottleneck of this algorithm is the sorting operation. Hence, the overall time complexity of this algorithm is $O(|\mathcal{P}| \log |\mathcal{P}|)$.

Theorem 1: We have $\Gamma(\mathcal{A} \cup \overline{\mathcal{P}}) \geq \Gamma(\mathcal{A}' \cup \overline{\mathcal{P}})$, where $\mathcal{A}$ from Algorithm 2 and $\mathcal{A}'$ is any subset of $\mathcal{P}$ with $\mathcal{A}' \neq \mathcal{A}$.

Theorem 1 shows that the result obtained from our Algorithm 2 is a global optimal solution to (7).

5) Convergence of Adaptive Pruning: As adaptive pruning can both increase and decrease the model size over time, a natural question is whether the model parameter vector will converge to a fixed value in a regular FL procedure with $I \geq 1$ local SGD iterations in each round. We study this problem in the following.

We first make the following minimal set of assumptions that are common in the literature [22], [23].

Assumption 1:

1) *Smoothness:*
$$\|\nabla F_n(\mathbf{w}_1) - \nabla F_n(\mathbf{w}_2)\| \leq \beta \|\mathbf{w}_1 - \mathbf{w}_2\| \quad \forall n, \mathbf{w}_1, \mathbf{w}_2$$
where β is a positive constant.
2) *Lipschitzness:*
$$\|F(\mathbf{w}_1) - F(\mathbf{w}_2)\| \leq L \|\mathbf{w}_1 - \mathbf{w}_2\| \quad \forall \mathbf{w}_1, \mathbf{w}_2$$
where L is a positive constant.
3) *Unbiasedness:*
$$\mathbb{E}\big[\mathbf{g}_n(\mathbf{w})\big] = \nabla F_n(\mathbf{w}) \quad \forall n, \mathbf{w}.$$

4) *Bounded Variance:*

$$\mathbb{E}\|\mathbf{g}_n(\mathbf{w}) - \nabla F_n(\mathbf{w})\|^2 \leq \sigma^2 \quad \forall n, \mathbf{w}.$$

5) *Bounded Divergence:*

$$\|\nabla F(\mathbf{w}) - \nabla F_n(\mathbf{w})\|^2 \leq \epsilon^2 \quad \forall n, \mathbf{w}$$

where $F(\mathbf{w}) := \sum_{n=1}^{N} p_n F_n(\mathbf{w})$, as defined in (1).

6) *Time Independence in SGD:* The stochastic gradients obtained in different iterations are independent of each other.

7) *Client Independence:* The stochastic gradients obtained from different clients are always independent of each other, even in the same iteration.

Theorem 2: When Assumption 1 holds, and $\eta \leq (1/(2\sqrt{6}I\beta))$, we have

$$\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\left\|\nabla F\left(\mathbf{w}'(k)\right)\odot \mathbf{m}_{\mathbf{w}'}(k)\right\|^2 \leq \frac{2(F_0 - F^*)}{\eta K} + \alpha\eta\beta\sigma^2 + 4\beta^2\left((1-\alpha)I\sigma^2 + 3I^2\epsilon^2\right)\eta^2 + \frac{2L}{\eta K}\sum_{k=0}^{K-1}\mathbb{E}\left\|\mathbf{w}(k) - \mathbf{w}'(k)\right\| \quad (8)$$

where $\alpha := \sum_{n=1}^{N} p_n^2$, $F_0 := F(\mathbf{w}(0))$, $F^* := \min_{\mathbf{w}} F(\mathbf{w})$, and I is the number of iterations per round.

Under some conditions, this convergence can be bounded asymptotically by $\mathcal{O}(1/\sqrt{NK}) + \mathcal{O}(1/K)$, achieving linear speedup[1] with the number of clients N [31], [45] for sufficiently large K. See Appendix B2 for more details.

If we do not reconfigure in an iteration k, we have $\mathbf{w}(k) = \mathbf{w}'(k)$. In the right-hand side (RHS) of (8), the first two terms go to zero as $K \to \infty$. The last term is related to how well $\mathbf{w}'$ approximates $\mathbf{w}$ after pruning. To ensure that the sum in the last term grows slower than $\sqrt{K}$, the number of *nonzero* prunable parameters (which belong to $\mathcal{P}$) should decrease over time. Note that we consider all zero parameters to be prunable, and they also belong to $\mathcal{P}$; thus, the size of $\mathcal{P}$ itself may not decrease over time. This convergence result shows that the gradient components corresponding to the remaining (i.e., not pruned) parameters vanish over time, which suggests that we will get a "stable" parameter vector in the end, because, when the gradient norm is small, the change of parameters in each iteration is also small. In addition to gradient convergence on the subspace after pruning, as suggested in Theorem 2, our experiments show that our pruned model also converges to an accuracy close to that of the full-sized model.

6) Tracking a Small Model: By choosing the size of $\overline{\mathcal{P}}$ properly over time, our adaptive pruning algorithm can keep reducing the model size as long as such reduction does not adversely impact further training. Intuitively, the model that we obtain from this process is one that has a small size while maintaining full "trainability" in future iterations. Parameter components for which the corresponding gradient components remain zero (or close to zero) will be pruned.

[1]The dominant term is $\mathcal{O}(1/\sqrt{NK})$ when K is sufficiently large. The notion of linear speedup means that, to reach the same error bound, the total number of rounds K can proportionally decrease as N increases.

In cases where a target maximum model size should be reached at convergence (e.g., for efficient inference later), we can also enforce a maximum size constraint in each reconfiguration that starts with the full size and gradually decreases to the target size as training progresses, which allows the model to train quickly in initial rounds while converging to the target size in the end.

V. Implementation

A. Using Sparse Matrices

Although the benefit of model pruning in terms of computation is constantly mentioned in the literature from a theoretical point of view [13], most existing implementations substitute sparse parameters by applying binary masks to dense parameters. Applying masks increases the overhead of computation, instead of reducing it. We implement sparse matrices for model pruning, and we show its efficacy in our experiments. We use dense matrices for full-sized models and sparse matrices for weights in both convolutional and fully connected layers in pruned models.

B. Complexity Analysis

1) Storage, Memory, and Communication: We implement two types of storage for sparse matrices: bitmap and value-index tuple. Bitmap uses one extra bit to indicate whether the specific value is zero. For 32-bit floating point parameter components, bitmap incurs $1/32$ extra storage and communication overhead. Value-index tuple stores the values and both row and column indices of all nonzero entries. In our implementation, we use 16-bit integers to store row and column indices and 32-bit floating-point numbers to store parameter values. Since each parameter component is associated with a row index and a column index, the storage and communication overhead doubles compared to storing the values only. We dynamically choose between the two ways of storage, and thus, the ratio of the sparse parameter size to the dense parameter size is $\min\{2 \times d, (1/32) + d\}$, where d is the model's *density* (percentage of nonzero parameters). This ratio is further optimized when the matrix sparsity pattern is fixed (in most FL rounds; see Appendix C). In this case, there is no extra cost since only values of the nonzero entries need to be exchanged.

2) Computation: Because dense matrix multiplication is extremely optimized, sparse matrices will show an advantage in computation time only when the matrix is below a certain density, where this density threshold depends on specific hardware and software implementations. In our implementation, we choose either dense or sparse representation depending on which one is more efficient. The complexity (computation time) of the matrix multiplication between a sparse matrix $\mathbf{S}$ and a dense matrix $\mathbf{D}$ is linear to the number of nonzero entries in $\mathbf{S}$ (assuming that $\mathbf{D}$ is fixed).

C. Implementation Challenges

As of today, well-known machine learning frameworks have limited support for sparse matrix computation. For instance, in PyTorch version 1.6.0, the persistent storage of a matrix in sparse form takes $5\times$ space compared to its dense form; the computations on sparse matrices are slow; sparse matrices are not supported for the kernels in convolutional layers; and so on. To benefit from using sparse matrices in real systems,

we extend the PyTorch library by implementing a more efficient sparse storage and the support for sparse convolutional kernels. We only partially improve backward passes due to implementation limitations (more details in Appendix C2). This problem, however, can be improved in the future by implementing and further optimizing efficient sparse matrix multiplication on low-level software, as well as developing specific hardware for this purpose. Nevertheless, the novelty in our implementation is that we use sparse matrices in both fully connected and convolutional layers in the pruned model.

VI. Experiments

In this section, we present the experimental setup and results.

Datasets: We evaluate PruneFL on four image classification tasks.

1) Conv-2 model on FEMNIST [46].
2) VGG-11 model [47] on CIFAR-10 [48].
3) ResNet-18 model [49] on ImageNet-100 [50].
4) MobileNetV3-Small model [51] on CelebA [46].

All of which represent typical FL tasks. Due to practical considerations of edge devices' training time and storage capacity, we select data corresponding to 193 writers for FEMNIST and the first 100 classes of the ImageNet dataset (referred to as ImageNet-100). We adapt some layers in VGG-11, ResNet-18, and MobileNetV3-Small to match the number of output labels in our datasets.

When using full client participation, because we only have ten clients in total, for FEMNIST, we partition all the 193 writers' images into ten clients (in the first nine clients, each has 19 writers' images, and the last client has 22 writers' images). For CelebA, we partition all the 9343 persons' images into ten clients (in the first nine clients, each has 934 persons' images, and the last client has 937 persons' images). Note that such partitioning is still non-IID.

Model Architectures: The architecture details are presented in Table C.1 in the appendix. VGG-11 ResNet-18, and MobileNetV3-Small are well-known architectures, and we directly acquire Conv-2 from its original work [46].

Platform: To study the performance of our proposed approach, we conduct experiments in: 1) a real edge computing prototype, where a personal computer serves as both the server and a client, and the other clients are Raspberry Pi devices and 2) a simulated setting with multiple clients and a server, where computation and communication times are obtained from measurements involving either Raspberry Pi devices or Android phones.

Unless otherwise specified, the prototype system includes nine Raspberry Pi (version 4, with 2-GB RAM and 32-GB SD card) devices as clients and a personal computer without GPU as both a client and the server (totaling ten clients). Three of the Raspberry Pis use wireless connections, and the remaining six use wired connections. The communication speed is stable and is approximately 1.4 MB/s. The simulated system uses the same setting as in the prototype. We use time measurements from Raspberry Pis, except for the ImageNet-100 dataset, where we replace the computation time with measurements from the Android virtual machine (VM).

We consider FL with full client participation in the main article and present results with random client selection [42] in Appendix D2. The results are similar. FEMNIST and CelebA data are partitioned into clients in a non-IID manner according to writer/person identity, and CIFAR-10 and ImageNet-100 are partitioned into clients in an IID manner.

Baselines: We compare the test accuracy versus *time* curve of PruneFL with five baselines: 1) conventional FL [2]; 2) iterative pruning [13]; 3) online learning [9]; 4) SNIP [18]; and 5) SynFlow [17]. Because iterative pruning and SNIP cannot automatically determine the model size, we consider an enhanced version of these baselines that obtain the same model size as PruneFL at convergence. Additional baselines are also considered in Section VI-B.

Since our experiments try to minimize the training time using pruning, there is no direct way of comparing with the baselines that either are not specifically designed for pruning (the online learning baseline) or do not adapt the pruned model size (all other baselines). We compare with the baselines as follows. In every round, the online learning approach produces a model size for the next round, and we adjust the model accordingly while keeping each layer's density the same. To compare with SNIP, after the first round, we let SNIP prune the original model in a one-shot manner to the same density as the final model found by our adaptive pruning method and keep the architecture afterward. Similarly, to compare with SynFlow, we let SynFlow prune the model (before training) to the same density as the final model found by our adaptive pruning method and keep the architecture afterward. To compare with iterative pruning, we let the model be pruned with a fixed rate for 20 times (at an equal interval) in the first half of the total number of rounds such that the remaining number of parameter components equals that of the model found by our adaptive pruning method, and the pruning rate is equal across layers.

Pruning Configurations: The initial pruning stage is done on the personal computer client. We end the initial pruning stage either when the model size is "stable," or when it exceeds certain maximum number of iterations. We consider the model size as "stable" when its relative change is below 10% for five consecutive reconfigurations.

For adaptive pruning, to ensure convergence of the last term on the RHS of (8) in Theorem 2, we exponentially decrease the number of *nonzero* prunable parameters in $\mathcal{P}$ over rounds. We note that $\mathcal{P}$ includes both zero and nonzero parameters; hence, the size of $\mathcal{P}$ itself may not decrease. For a given size of $\mathcal{P}$, the $|\mathcal{P}|$ parameters with the smallest magnitude belong to $\mathcal{P}$ that can be pruned (or added back), and the rest belongs to $\overline{\mathcal{P}}$ that cannot be pruned.

Biases (if any) in the DNNs are not pruned. In ResNet-18, BatchNorm layers and downsampling layers are not pruned since the number of parameters in such layers is negligible compared to the size of convolutional and fully connected layers.

Lottery Ticket Analysis: To verify whether the final model from adaptive pruning is a lottery ticket [11], [14], we reinitialize this converged model using the original random seed and compare its accuracy versus *round* curve with: 1) conventional FL; 2) random reinitialization (same architecture as the lottery ticket but initialized with a different random seed); 3) SNIP; and 4) SynFlow.

TABLE II
EVALUATION CONFIGURATIONS (C.S. STANDS FOR CLIENT SELECTION; LR STANDS FOR LEARNING RATE)

Dataset	FEMNIST	CIFAR-10	ImageNet-100	CelebA
SGD params in round r	LR = 0.25	LR = $0.1 \cdot 0.5^{\frac{r}{10000}}$	LR = $0.05 \cdot 0.5^{\lfloor \frac{r}{1000} \rfloor \cdot 0.1}$	LR = 0.2
Fraction of non-zero prunable parameters in round r	$0.3 \cdot 0.5^{\frac{r}{10000}}$	$0.3 \cdot 0.5^{\frac{r}{10000}}$	$0.3 \cdot 0.5^{\frac{r}{10000}}$	$0.3 \cdot 0.5^{\frac{r}{10000}}$
Number of data samples used in initial pruning	200	200	500	500
Number of clients (Non-C.S., C.S.)	10, 193	10, 100	10, 100	10, 934
Mini-batch size, local iterations in each round	20, 5	20, 5	20, 5	20, 5
Reconfiguration	every 50 rounds	every 50 rounds	every 50 rounds	every 50 rounds
Total number of FL rounds	10,000	10,000	20,000	1,000
Evaluation	prototype (Pi 4), simulation (Pi 4)	simulation (Pi 4)	simulation (Android VM)	simulation (Pi 4)

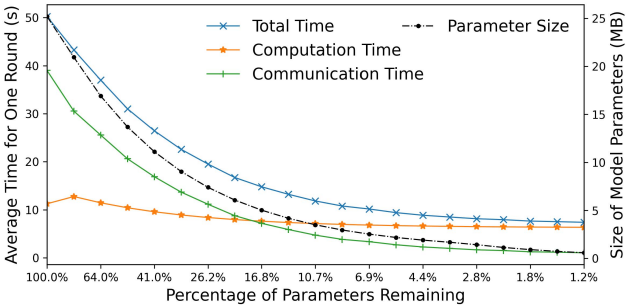


Fig. 2. Training time on Raspberry Pi 4 (FEMNIST).

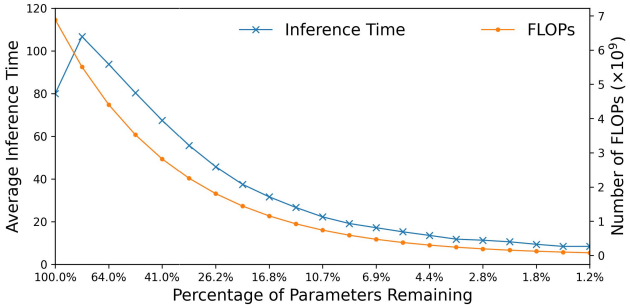


Fig. 3. Inference time on Raspberry Pi 4 (FEMNIST).

Hyperparameters: The hyperparameters above are chosen empirically only with coarse tuning by experience. We observe that our and other methods are insensitive to these hyperparameters. Hence, we do not perform fine-tuning on any parameter.

The detailed evaluation configurations are given in Table II.

A. Time Measurement

We present the time measurements of one FL round on the prototype system to show the effectiveness of model pruning on edge devices. We implement the full-sized Conv-2 model in dense form and the pruned models in the sparse form at different densities and measure the average elapsed time of FL on these pruned models involving both the server and clients over ten rounds.

Fig. 2 shows the average total time, computation time, and communication time in one round as we vary the model density. Note that the model is in the dense form at 100% on the x-axis and sparse form elsewhere. We also plot the actual size of the parameters that are exchanged between server and clients in this figure for one FL round.

1) Computation Time: We see from Fig. 2 that, as the model density decreases, the computation time (for five local iterations) decreases from 11.24 s per round to 6.34 s per round. This reduction in computation time is moderate since our implementation of sparse computation only partially improves backward passes (see Section V). In addition, we plot in Fig. 3 the total inference time and the number of floating-point operations (FLOPs) for 200 data samples (see Appendix C3 for details of FLOPs computation). The inference time result shows similar trends as in Fig. 2, and the number of FLOPs keeps decreasing as we reduce the model size.

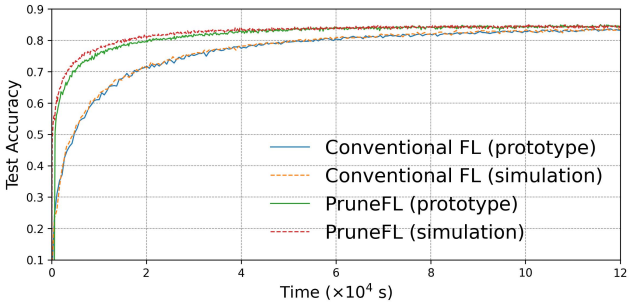


Fig. 4. Comparing conventional FL and PruneFL with both prototype and simulation results (FEMNIST).

2) Communication Time: Our implementation of sparse matrices reduces the storage requirement significantly (see Section V). Compared to the computation time, the decrease in the communication time is more noticeable. It drops from 35.88 s per round to 1.04 s per round.

3) Enabling FL on Low-Power Edge Device: In addition, we ran experiments with the LeNet-300-100 [52] architecture, and we observed that, when training the MNIST [52] dataset on the full-sized, dense-form LeNet-300-100 model on Raspberry Pi *version 3* (with 1-GB RAM and 32-GB SD card), the system dies during the first mini-batch due to resource exhaustion, while the models in sparse form can be trained. Thus, our approach of using sparse models enables model training on low-power edge devices, which is otherwise

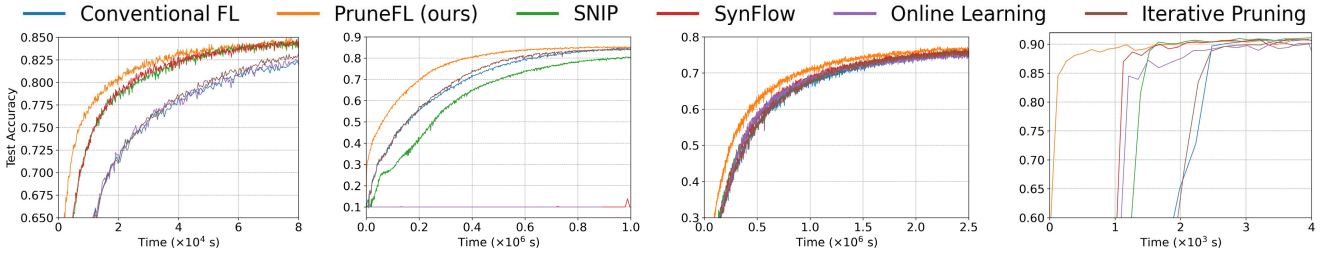


Fig. 5. Test accuracy versus time results of four datasets. (a) Conv-2 on FEMNIST. (b) VGG-11 on CIFAR-10. (c) ResNet-18 on ImageNet-100. (d) MobileNetV3-Small on CelebA.

impossible on Raspberry Pi 3 in this experiment due to the device's resource limitation.

B. Training Cost Reduction

In the following, we study PruneFL's cost reduction in terms of both time and FLOPs for training.

1) Comparing Conventional FL and PruneFL: Fig. 4 shows the test accuracy versus time results on both the prototype and simulated systems for Conv-2 on FEMNIST. The time for initial pruning of PruneFL is included in this figure, which is negligible (it takes less than 500 s) compared to the further pruning stage. We see that PruneFL outperforms conventional FL by a significant margin. Since the prototype and simulation results match closely, we present the simulation results in subsequent experiments due to their excessive training time on the prototype system.

2) Training Time Reduction: In Fig. 5, we compare the test accuracy versus time results for all datasets, models, and baselines. It is clear that PruneFL demonstrates a consistent advantage in training speed over baselines. Moreover, PruneFL always converges to similar accuracy achieved by conventional FL (see Appendix D3). Other methods may have suboptimal performance, e.g., SNIP does not converge to conventional FL's accuracy with CIFAR-10. We also observe that some approaches, such as online learning and SynFlow in Fig. 5(b), always stay at the random guess accuracy. The reason could be that such approaches are unstable and can get stuck in local optimal points at the beginning of training.

3) Training FLOPs' Reduction: Although we observe that the training time on Raspberry Pis is relatively consistent across different models and tasks, there are still factors that can affect the training time (e.g., environment temperature). To further validate our approach's advantage in accelerating training, we present the results on test accuracy versus accumulated FLOPs per client for FEMNIST in Fig. 6. We find that this result shares the same characteristics with Fig. 5(a) in terms of acceleration (we present only one set of results here due to the similarity).

4) Time and FLOPs to Reach Target Accuracy: Table III lists the time and accumulated FLOPs per client that an algorithm first reaches a certain accuracy with FEMNIST. PruneFL takes less than $1/3$ of time compared to conventional FL to reach 80% accuracy, and it also saves more than 33% of time (more than 2 h) compared to SNIP and SynFlow. The savings of FLOPs are similar.

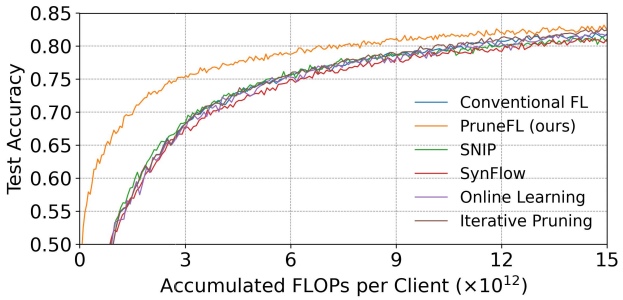


Fig. 6. Test accuracy versus accumulated FLOPs per client (FEMNIST).

TABLE III
TIME AND ACCUMULATED FLOPS PER CLIENT TO REACH TARGET ACCURACY (FEMNIST)

Approach	Time (FLOPs) to reach 70% accuracy	Time (FLOPs) to reach 80% accuracy
Conventional FL	17,929 s (3.5 TFLOPs)	52,153 s (10.5 TFLOPs)
PruneFL (ours)	**3,187 s (1.6 TFLOPs)**	**15,009 s (6.8 TFLOPs)**
SNIP	6,801 s (3.3 TFLOPs)	22,467 s (11.7 TFLOPs)
SynFlow	7,132 s (3.6 TFLOPs)	22,327 s (12.3 TFLOPs)
Online	18,042 s (3.5 TFLOPs)	54,593 s (10.7 TFLOPs)
Iterative	17,495 s (3.5 TFLOPs)	46,521 s (10.1 TFLOPs)

5) Comparing With Additional Baselines: To avoid bottlenecks, our algorithm and implementation ensure that all components in PruneFL, including communication, computation, and reconfiguration, are orchestrated and inexpensive. For this reason, some approaches in the literature that is not specifically designed for the edge computing environment with low-power devices may perform poorly if applied to our system setup, as we illustrate next.

Considering computation time, PruneTrain [21] applies regularization on every input channel and output channel in every layer. When the same regularization is applied to our system, we find that the computation time (using FEMNIST and Conv-2) takes 17.65 s per round, which is a 57% increase compared to PruneFL.

Considering communication time, dynamic pruning with feedback (DPF) [19] maintains a full-sized model, and clients have to upload full-sized gradients to the server (but only download a subset of model parameters) in every round. Thus, assuming unit model size and model density d, the communication cost per round, including both uploading and downloading, is $1+d$. In comparison, clients in PruneFL only upload the full-sized model to the server at a reconfiguration

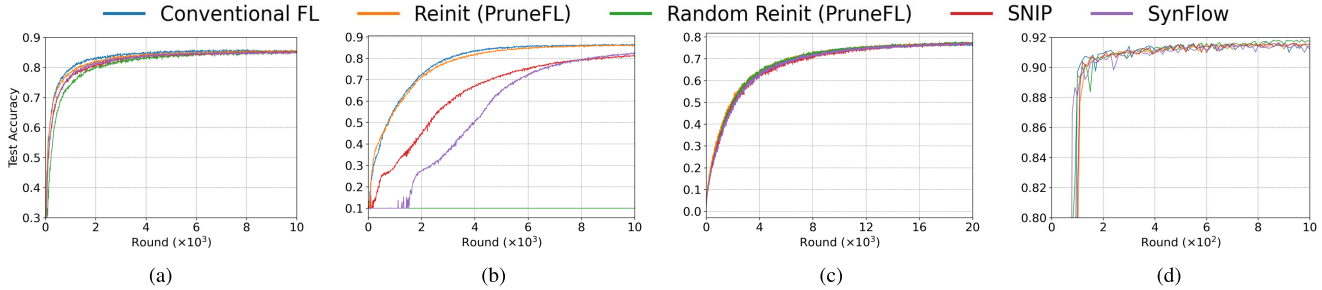


Fig. 7. Lottery ticket results of four datasets. (a) Conv-2 on FEMNIST. (b) VGG-11 on CIFAR-10. (c) ResNet-18 on ImageNet-100. (d) MobileNetV3-Small on CelebA.

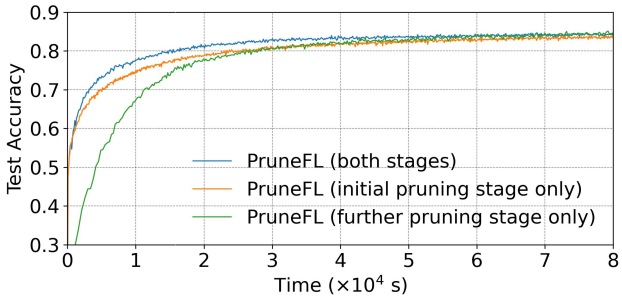


Fig. 8. PruneFL with only one pruning stage (FEMNIST).

round (every 50 rounds in our experiments) and always exchange pruned models otherwise. This gives an average cost of $(((1+d)+2\times 49d)/50) = 0.02 + 1.96d$, including both uploading and downloading. For instance, when the model density is 10%, DPF incurs 5.1× communication cost compared to PruneFL. Finally, our reconfiguration algorithm (see Algorithm 2) runs in quasi-linear time, making it possible to be implemented on edge devices.

C. Finding a Lottery Ticket

Unlike some existing pruning techniques, such as SNIP [18], dynamic pruning [19], and SynFlow [17], PruneFL finds a lottery ticket (although not necessarily the smallest). In Fig. 7, FL with the reinitialized pruned model obtained from PruneFL learns comparably fast as FL with the original model, in terms of test accuracy versus *round*, confirming that they are lottery tickets. In Fig. 7(b), the Random Reinit curve stays at the random guess accuracy. This is not surprising since the lottery ticket, i.e., the final pruned model found by PruneFL, needs to be reinitialized to its original values to learn comparably fast as the full-sized model [11]. When reinitialized with different values, the training of the "lottery ticket" can be suboptimal. In this experiment, it is stuck at the beginning of training.

Fig. 8 compares the test accuracy versus round curves of PruneFL with alternative methods that either only includes initial pruning (at a single client) or only includes further pruning (during FL). It shows that the model obtained from the initial pruning stage does not converge to the optimal accuracy and only performing further pruning without initial pruning results in a slower learning speed. PruneFL with both stages avoids drawbacks from methods that include only one stage. Furthermore, the model obtained from initial pruning is not a lottery ticket of the original model, while PruneFL with only further pruning or both pruning stages finds a lottery ticket.

Therefore, one can view PruneFL as a two-stage procedure to find a lottery ticket of the given model, which is in line with our claims in Section IV. The ability that we can find lottery tickets is useful when we need to retrain a pruned model on slightly different but similar datasets [14].

D. Model Size Adaptation

An illustration of the change in model size is shown in Fig. 9. The small negative part of the x-axis shows the initial pruning stage, which is unique to PruneFL. Since there is no notion of "round" in the initial pruning stage, we consider five local iterations in this stage as one round, which is consistent with our FL setting. Conventional FL always keeps the full model size. The model sizes provided by online learning are unstable. It fluctuates in initial rounds due to its exploration. SNIP and SynFlow prune the initial model to the target size in a one-shot manner at the beginning of training. Iterative pruning gradually reduces the model size until reaching the target. We notice that PruneFL also discovers the degree of overparameterization. Empirically, Conv-2, an overparameterized model for FEMNIST, converges to a small density (13.4%), while ResNet-18, an underparameterized model for ImageNet-100, converges to a density of around 67.7%.

It is worth mentioning that finding a proper target density for pruning is nontrivial. Usually, foresight pruning methods, such as [18] and [43], prune the model to the (manually selected) density before training. Fig. 10 shows two cases where we use SNIP to prune Conv-2 (with FEMNIST) to 30% and 1%, and the training speed becomes slower; if the density is too small (1%), the sparse model cannot converge to the same accuracy as the original model. In comparison, PruneFL automatically determines a proper density.

E. Training With Limited/Targeted Model Sizes

There are cases where a hard limit on the maximum model size or a targeted final model size (or both) is desired. For example, if some of the client devices have limited memory or storage so that only a partial model can be loaded, then the model size must be constrained after initial pruning. Targeted model size may be needed in the case where the goal of the FL system is to obtain a model with a certain small size at the end of training.

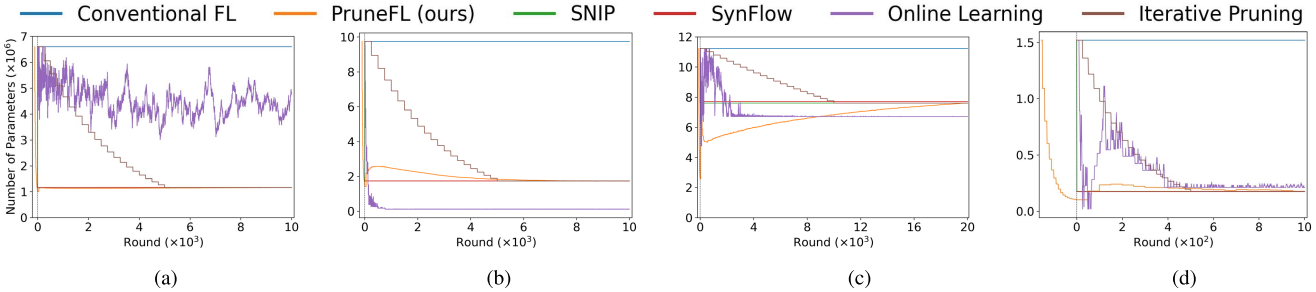


Fig. 9. Number of parameters versus round for four datasets. (a) Conv-2 on FEMNIST. (b) VGG-11 on CIFAR-10. (c) ResNet-18 on ImageNet-100. (d) MobileNetV3-Small on CelebA.

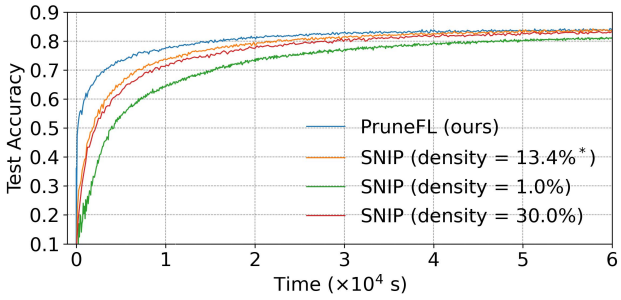


Fig. 10. SNIP with different densities (FEMNIST).

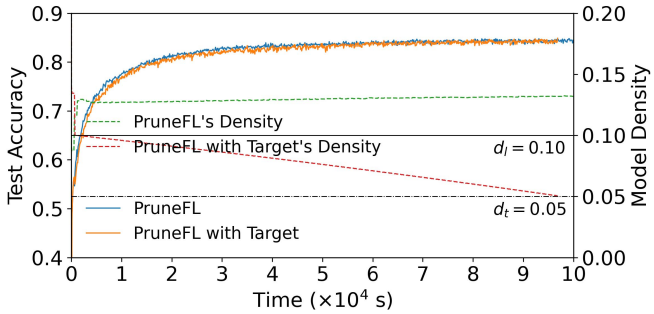


Fig. 11. Training with limited and targeted size (FEMNIST).

Next, we present an extended PruneFL with limited and targeted model sizes, and show that, with reasonable constraints, PruneFL still achieves good results. We use a heuristic way to limit the model size: we stop Algorithm 2 early when the number of remaining parameters reaches the maximum allowed size, and we schedule the maximum size of the model to decrease linearly. Assuming that d_l is the density limit, d_t is the target model density at the end of the further pruning stage ($d_t \leq d_l$), and PruneFL is run for $r_{\max}$ rounds after the initial pruning stage, then the maximum density at round r is $d_{\max}(r) = (r \cdot d_t + (r_{\max} - r) \cdot d_l)/r_{\max}$. The results of selecting $d_l = 10\%$ and $d_t = 5\%$ for Conv-2 on FEMNIST are given in Fig. 11. We see that, if we do not impose these model size constraints, PruneFL exceeds the density limits $d_l = 10\%$ and $d_t = 5\%$ defined in this example and obtains a model that is much larger than the target density $d_t = 5\%$ at the end of training. We see that PruneFL with effective size limit and target still achieves fast convergence and similar convergence accuracy, and the model size is always limited below the threshold $d_l = 10\%$ and reaches the target density $d_t = 5\%$ at the end of training.

TABLE IV
DENSITY OF EACH LAYER AT CONVERGENCE (NO C.S)

Experiment	VGG-11 on CIFAR-10
Convolutional	1.0, 0.80, 0.84, 0.84, 0.50, 0.07, 0.01, 0.03
Fully-connected	0.14, 0.15, 0.7

F. Relative Importance Between Layers

Similar to SynFlow [17] and SNIP [18], our algorithm discovers the relative importance between layers automatically. Taking CIFAR-10 on VGG-11 as an example, the densities of convolutional layers and fully connected layers at convergence are listed in Table IV in a sequential manner. We observe that the input and output layers are not pruned to a low density, indicating that they are relatively important in the neural network architectures. This also agrees with the pruning scheme in [11], where the authors empirically set the pruning rate of the output layer to a small percentage or even to zero. Some large convolutional layers, such as the last two convolutional layers in VGG-11, are identified as redundant and, thus, have small densities at convergence.

VII. CONCLUSION

We have proposed PruneFL for FL in edge/mobile computing environments, where the goal is to effectively reduce the size of neural networks so that resource-limited clients can train them within a short time. Our PruneFL method includes initial and further pruning stages, which improves the performance compared to only having a single stage. PruneFL also includes a low-complexity adaptive pruning method for efficient FL, which finds the desired model size that can achieve a similar prediction accuracy as the original model but with much less time. Our experiments on Raspberry Pi devices confirm that we improve the cost-efficiency of FL while obtaining a lottery ticket. Our method can be applied together with other compression techniques, such as quantization, to further reduce the communication overhead.

ACKNOWLEDGMENT

The views and conclusions contained in this article are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Office of Naval Research, the U.S. National Science Foundation, the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence, or the U.K. Government. The

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

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

[3] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," 2019, *arXiv:1908.07873*.

[4] J. Park, S. Samarakoon, M. Bennis, and M. Debbah, "Wireless network intelligence at the edge," *Proc. IEEE*, vol. 107, no. 11, pp. 2204–2239, Nov. 2019.

[5] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, p. 12, 2019.

[6] P. Kairouz *et al.*, "Advances and open problems in federated learning," 2019, *arXiv:1912.04977*.

[7] 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*.

[8] S. Caldas, J. Konečny, H. B. McMahan, and A. Talwalkar, "Expanding the reach of federated learning by reducing client resource requirements," 2018, *arXiv:1812.07210*.

[9] P. Han, S. Wang, and K. K. Leung, "Adaptive gradient sparsification for efficient federated learning: An online learning approach," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Nov. 2020, pp. 300–310.

[10] Z. Xu, Z. Yang, J. Xiong, J. Yang, and X. Chen, "ELFISH: Resource-aware federated learning on heterogeneous edge devices," 2019, *arXiv:1912.01684*.

[11] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)* New Orleans, LA, USA, May 2019, pp. 6–9. [Online]. Available: https://dblp.org/rec/conf/iclr/FrankleC19.bib

[12] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 598–605.

[13] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[14] A. Morcos, H. Yu, M. Paganini, and Y. Tian, "One ticket to win them all: Generalizing lottery ticket initializations across datasets and optimizers," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 4933–4943.

[15] S. Narang, E. Elsen, G. Diamos, and S. Sengupta, "Exploring sparsity in recurrent neural networks," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)* Toulon, France, Apr. 2017, pp. 24–26. [Online]. Available: https://openreview.net/forum?id=BylSPv9gx

[16] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," 2017, *arXiv:1710.01878*.

[17] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 6377–6389. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/46a4378f835dc8040c8057beb6a2da52-Paper.pdf

[18] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)* New Orleans, LA, USA, May 2019, pp. 6–9. [Online]. Available: https://openreview.net/forum?id=B1VZqjAcYX

[19] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi, "Dynamic model pruning with feedback," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)* Addis Ababa, Ethiopia, Apr. 2020, pp. 26–30. [Online]. Available: https://openreview.net/forum?id=SJem8lSFwB

[20] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, pp. 1–18, 2017.

[21] S. Lym, E. Choukse, S. Zangeneh, W. Wen, S. Sanghavi, and M. Erez, "PruneTrain: Fast neural network training by dynamic sparse model reconfiguration," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2019, pp. 1–13.

[22] H. Yu, S. Yang, and S. Zhu, "Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 5693–5700.

[23] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of FedAvg on non-IID data," in *Proc. 8th Int. Conf. Learn. Represent.(ICLR)* Addis Ababa, Ethiopia, Apr. 2020, pp. 26–30. [Online]. Available: https://openreview.net/forum?id=HJxNAnVtDS

[24] J. Wang, S. Wang, R.-R. Chen, and M. Ji, "Local averaging helps: Hierarchical federated learning and convergence analysis," 2020, *arXiv:2010.12998*.

[25] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD," in *Proceedings of Machine Learning and Systems*, vol. 1, A. Talwalkar, V. Smith and M. Zaharia Eds., 2019, pp. 212–229. [Online]. Available: https://proceedings.mlsys.org/paper/2019/file/c8ffe9a587b126f152ed3d89a146b445-Paper.pdf

[26] S. Wang *et al.*, "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 1205–1221, Jun. 2019.

[27] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning," 2019, *arXiv:1910.06378*.

[28] S. P. Karimireddy, Q. Rebjock, S. Stich, and M. Jaggi, "Error feedback fixes SignSGD and other gradient compression schemes," in *Proc. Int. Conf. Mach. Learn.*, vol. 97, Jun. 2019, pp. 3252–3261.

[29] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The convergence of sparsified gradient methods," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 5973–5983.

[30] S. Shi, K. Zhao, Q. Wang, Z. Tang, and X. Chu, "A convergence analysis of distributed SGD with communication-efficient gradient sparsification," in *Proc. 28th Int. Joint Conf. Artif. Intell. (IJCAI)*, Aug. 2019, pp. 3411–3417.

[31] P. Jiang and G. Agrawal, "A linear speedup analysis of distributed deep learning with sparse and quantized communication," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 2525–2536.

[32] W. Du, X. Zeng, M. Yan, and M. Zhang, "Efficient federated learning via variational dropout," Tech. Rep., 2018.

[33] A. Li *et al.*, "LotteryFL: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-IID datasets," 2020, *arXiv:2008.03371*.

[34] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.

[35] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proc. Brit. Mach. Vis. Conf.*, 2014, doi: 10.5244/C.28.88.

[36] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[37] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2178–2188.

[38] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 1423–1431.

[39] 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*.

[40] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1175–1191.

[41] T. Li, M. Sanjabi, A. Beirami, and V. Smith, "Fair resource allocation in federated learning," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)* Addis Ababa, Ethiopia, Apr. 2020, pp. 26–30. [Online]. Available: https://openreview.net/forum?id=ByexElSYDr

[42] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," 2019, *arXiv:1902.01046*.

[43] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)* Addis Ababa, Ethiopia, Apr. 2019, pp. 26–30. [Online]. Available: https://openreview.net/forum?id=SkgsACVKPH

[44] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2016, *arXiv:1611.06440*.

[45] H. Yu, R. Jin, and S. Yang, "On the linear speedup analysis of communication efficient momentum SGD for distributed non-convex optimization," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7184–7193.

[46] S. Caldas *et al.*, "LEAF: A benchmark for federated settings," 2018, *arXiv:1812.01097*.

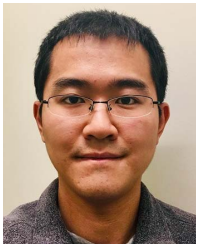
[47] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[48] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[50] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.
[51] A. Howard *et al.*, "Searching for MobileNetV3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1314–1324.
[52] 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.

Yuang Jiang received the B.S. degree in applied physics and the B.E. degree in computer science from the University of Science and Technology of China, Hefei, Anhui, China, in 2016. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, Yale University, New Haven, CT, USA.

His Ph.D. research focuses on the design and implementation of resource allocation algorithms in networked systems.



Shiqiang Wang (Member, IEEE) received the Ph.D. degree from the Department of Electrical and Electronic Engineering, Imperial College London, London, U.K., in 2015.

He has been a Research Staff Member with the IBM T. J. Watson Research Center, NY, USA, since 2016. His current research focuses on the intersection of distributed computing, machine learning, networking, and optimization, with a broad range of applications including data analytics, edge-based artificial intelligence (Edge AI), the Internet of Things (IoT), and future wireless systems.

Dr. Wang received the IEEE Communications Society (ComSoc) Leonard G. Abraham Prize in 2021, the IEEE ComSoc Best Young Professional Award in Industry in 2021, the IBM Outstanding Technical Achievement Awards (OTAA) in 2019 and 2021, multiple Invention Achievement Awards from IBM since 2016, the Best Paper Finalist of the IEEE International Conference on Image Processing (ICIP) 2019, and the Best Student Paper Award of the Network and Information Sciences International Technology Alliance (NIS-ITA) in 2015. He serves as an Associate Editor for the IEEE TRANSACTIONS ON MOBILE COMPUTING, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and the IEEE TRANSACTIONS ON COMPUTATIONAL SOCIAL SYSTEMS.

Víctor Valls received the bachelor's degree in engineering and the M.Sc. degree in telecommunications from Universitat Pompeu Fabra, Barcelona, Spain, in 2011 and 2012, respectively, and the Ph.D. degree from Trinity College Dublin, Dublin, Ireland, in 2017.

He is currently a Marie Skłodowska-Curie Fellow with Trinity College Dublin and Yale University, New Haven, CT, USA. His research interests are in mathematical optimization and its applications to networks, control, and machine learning.

Bong Jun Ko received the B.S. and M.S. degrees from Seoul National University, Seoul, South Korea, in 1994 and 1996, respectively, and the Ph.D. degree in electrical engineering from Columbia University, New York, NY, USA, in 2005.

He would like to drive innovation drawn from a wide range of technical domains such as AI/ML, cloud computing, data analytics, and IoT, and also from a historical perspective on how technological innovations have helped shape modern society and improve standards of living. He was with Samsung. He was an AI Engineering Fellow with the Stanford Human-Centered AI Institute, worked as a Research Scientist with IBM T. J. Watson Research Center, and co-founded NeoMTel that pioneered image- and video-based services in the text-based mobile era. He was with Samsung. He is currently an Executive Vice President of Engineering at Samsung Electronics, Visual Display Division, where he leads the efforts to create and develop various software services on Samsung's global footprint of screen platforms.

Wei-Han Lee received the bachelor's degree in physics and electrical engineering from National Taiwan University, Taipei, Taiwan, in 2012, and the Ph.D. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2019.

He is currently a Research Staff Member with the IBM T.J. Watson Research Center. His current research interest includes AI security, big data privacy, and distributed AI.

Kin K. Leung (Fellow, IEEE) received the B.S. degree from the Chinese University of Hong Kong, Hong Kong, in 1980, and the M.S. and Ph.D. degrees from the University of California at Los Angeles, California, CA, USA, in 1982 and 1985, respectively.

He began his career at AT&T Bell Labs in New Jersey in 1986. Since 2004, he has been the Tanaka Chair Professor with Imperial College in London. His current research focuses on machine learning and optimization for communication, computer, and sensor networks. He also works on multiantenna systems for wireless networks.

Dr. Leung received the Distinguished Member of Technical Staff Award from AT&T Bell Labs in 1994. He received the Royal Society Wolfson Research Merits Award from 2004 to 2009 and became a member of Academia Europaea in 2012 and an IET Fellow in 2021. Jointly with his collaborators, he received the IEEE ComSoc Leonard G. Abraham Prize in 2021, the U.S.-U.K. Science and Technology Stocktake Award in 2021, the Lanchester Prize Honorable Mention Award in 1997, and several best paper awards. He chaired the IEEE Fellow Evaluation Committee for ComSoc from 2012 to 2015. He has served as an Editor for ten IEEE and ACM journals. He currently chairs the Steering Committee for the IEEE Transactions on Mobile Computing and is an Editor for the ACM COMPUTING SURVEY AND INTERNATIONAL JOURNAL ON SENSOR NETWORKS.

Leandros Tassiulas (Fellow, IEEE) received the Ph.D. degree in electrical engineering from the University of Maryland, College Park, MD, USA, in 1991.

He has held faculty positions with Polytechnic University, New York, NY, USA, the University of Maryland, the University of Ioannina, and the University of Thessaly, Thessaly, Greece. He is currently a John C. Malone Professor of electrical engineering with Yale University, New Haven, CT, USA. His research interests are in the field of computer and communication networks with emphasis on fundamental mathematical models and algorithms of complex networks, architectures and protocols of wireless systems, sensor networks, novel Internet architectures, and experimental platforms for network research. His most notable contributions include the max-weight scheduling algorithm and the back-pressure network control policy, opportunistic scheduling in wireless, the maximum lifetime approach for wireless network energy management, and the consideration of joint access control and antenna transmission management in multiple antenna wireless systems.

Dr. Tassiulas is a fellow of ACM in 2020. His research has been recognized by several awards including the IEEE Koji Kobayashi Computer and Communications Award in 2016, the ACM SIGMETRICS Achievement Award 2020, The inaugural INFOCOM 2007 Achievement Award for fundamental contributions to resource allocation in communication networks, several best paper awards, including the INFOCOM 1994, 2017, and Mobihoc 2016, a National Science Foundation (NSF) Research Initiation Award in 1992, an NSF CAREER Award in 1995, an Office of Naval Research Young Investigator Award in 1997, and a Bodossaki Foundation Award in 1999.