Model Pruning Enables Efficient Federated Learning on Edge Devices

Yuang Jiang[1*], **Shiqiang Wang**[2*], **Víctor Valls**[1], **Bong Jun Ko**[3],
Wei-Han Lee[2], **Kin K. Leung**[4], **Leandros Tassiulas**[1]
[1]Yale University [2]IBM Research [3]Stanford HAI [4]Imperial College London

Abstract

A challenge in federated learning (FL) is that client devices in FL usually have much more limited computation and communication resources compared to servers in a datacenter. 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: (i) we significantly reduce the training time compared to conventional FL and various other pruning-based methods; (ii) the pruned model converges to an accuracy that is very similar to the original model but has a much smaller size, and it is also a lottery ticket of the original model.

1 Introduction

The past decade has seen a rapid development of machine learning algorithms and applications, particularly in the area of deep neural networks (DNNs) [7]. However, a huge volume of training data is usually required to train accurate models for complex tasks. 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 of distributed model training from decentralized data [12, 20, 25, 28, 34]. In a typical FL system, data is collected by client devices (e.g., smartphones, cameras, smart sensors) at the network edge; the training process includes (i) local model updates using each client's own data and (ii) the fusion of all clients' models typically through a server. In this way, the data remains local in clients.

Client devices in FL are usually much more resource-constrained than server machines, in terms of computation power, communication bandwidth, memory and storage size, etc. Training DNNs that can include over millions of parameters (weights) on such resource-limited edge devices can take prohibitively long and consume a large amount of energy. In addition, clients may have intermittent availability which can further prolong the training process. 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 towards this direction recently using model/gradient compression techniques, where instead of training the original model with 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 [3, 8, 15, 33]. However, the former approach may reduce the accuracy of the final model in undesirable ways, whereas the latter approach only reduces the

[*]*Contact authors:* Yuang Jiang (yuang.jiang@yale.edu), Shiqiang Wang (wangshiq@us.ibm.com)
This work was presented at the NeurIPS Workshop on Scalability, Privacy, and Security in Federated Learning (SpicyFL) in 2020. For the latest version of this work, please see `https://arxiv.org/abs/1909.12326`.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

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.

In this paper, 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.

- **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 prunes at a single client only or directly involves multiple FL clients for pruning, especially when the clients have heterogeneous computational power.
- **Adaptive pruning.** PruneFL continuously "tracks" a model that is small enough for efficient transmission and computation with low memory 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 model accuracies. 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 of the original model as we show experimentally.
- **Implementation.** We implement FL with model pruning on real edge devices, where we extend a deep learning framework to support efficient sparse matrix computation.

2 Related Work

Neural network pruning. To reduce the complexity of neural network models, different ways of iterative parameter pruning that is interleaved with the model training process were proposed in the literature [9, 18, 24, 27, 36]. In addition to iterative pruning methods, an approach of one shot pruning at neural network initialization (before training) was proposed in SNIP [19]. A dynamic pruning approach that allows the network to grow and shrink during training was proposed in [23]. These methods [19, 23], however, do not conform to the lottery ticket hypothesis [6, 26]. The lottery ticket is useful for retraining a pruned model on a different yet similar dataset [26]. The above 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.

Efficient federated learning. To improve the communication efficiency of FL, methods for adapting or optimizing the communication frequency were studied [13, 31, 32]. An approach of parameter averaging using structured, sketched, and quantized updates was introduced in [15], which belongs to the broader area of gradient compression/sparsification [1, 11, 14, 29], where the degree of sparsity or compression is configured as a hyperparameter. An online learning approach that determines a near-optimal gradient sparsity was proposed in [8], which includes exploration steps that may slow down the training initially. To reduce both communication and computation costs, efficient FL techniques using lossy compression and dropout were developed [3, 33], where the final model still has the original size and hence providing no benefit for efficient inferencing. In addition, most existing studies on FL are based on simulation. Only a few recent papers considered implementation on real embedded devices [32, 33], which, however, do not include parameter pruning.

3 PruneFL

We consider an FL system with N clients. Each client $n \in [N] := \{1, 2, ..., N\}$ has a local empirical risk $F_n(\mathbf{w}) := \frac{1}{D_n} \sum_{i \in \mathcal{D}_n} f_i(\mathbf{w})$ 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, mean square error, etc.) 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 $F(\mathbf{w}) := \sum_{n \in [N]} p_n F_n(\mathbf{w})$, 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 set

$p_n = D_n/D$ with $\mathcal{D} := \bigcup_n \mathcal{D}_n$ and $D := |\mathcal{D}|$, we have $F(\mathbf{w}) = \frac{1}{D}\sum_{i\in\mathcal{D}} f_i(\mathbf{w})$. Other ways of configuring p_n may also be used to account for fairness [21].

The FL procedure usually involves multiple stochastic gradient descent (SGD) steps on the local empirical risk $F_n(\mathbf{w})$ computed by client n, followed by a parameter fusion step that involves the server collecting clients' local parameters and computing an aggregated parameter. In FedAvg [25], the aggregated parameter is simply the average of local parameters weighted by p_n. In the following, we call this procedure of multiple local SGD steps followed by a fusion step a *round*.

Our proposed PruneFL includes two stages: initial pruning at a selected client and further pruning involving both the server and clients during the FL process. We use *adaptive pruning* in both stages. In the following, we introduce the two pruning stages (Section 3.1) and adaptive pruning (Section 3.2).

3.1 Two-stage Distributed Pruning

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. (i) 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. (ii) 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. Initial pruning is done adaptively (Section 3.2) until the model size remains almost unchanged. Having only one client makes this pruning stage local, which requires no communication except for exchanging the original and final models. Nonetheless, the initial pruning stage can be directly extended to a group of clients. In that case, this stage is similar to a conventional FL process.

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. 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 makes the training most efficient. In this stage, data from all participating clients are involved. We will show later that in addition to speeding up training, our method also provides a lottery ticket of the original model.

3.2 Adaptive Pruning

Adaptive pruning includes both removing and adding back parameters, so we refer to such pruning operations as *reconfiguration*. We reconfigure the model at a given interval that includes multiple iterations. For further pruning, reconfiguration is done at the server after receiving parameter updates from clients (i.e., at the boundary between two FL rounds). In each reconfiguration step, adaptive pruning finds an optimal set of remaining (i.e., not pruned) model parameters for the most efficient training in the near future. Then, the parameters are pruned or added back accordingly, and the resulting model is used for training until the next reconfiguration.

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 after parameter averaging in FL, all clients start with the same parameter vector $\mathbf{w}$. Hence, we investigate the change of empirical risk after one SGD iteration starting with a common parameter $\mathbf{w}(k)$ in iteration k, for both initial and further pruning stages.

Definitions. Let $\mathbf{g}_{\mathbf{w}}(k)$ denote the stochastic gradient evaluated at $\mathbf{w}(k)$ and computed on the full parameter space in iteration k, such that $\mathbb{E}[\mathbf{g}_{\mathbf{w}}(k)] = \nabla F(\mathbf{w}(k))$. 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 $\odot$ is the element-wise product. When the model is pruned at the end of iteration k, parameter update in the next iteration will be done on the pruned parameter $\mathbf{w}'(k)$, so we have an SGD update step as:

$$\mathbf{w}(k+1) = \mathbf{w}'(k) - \eta \mathbf{g}_{\mathbf{w}'}(k) \odot \mathbf{m}_{\mathbf{w}'}(k). \tag{1}$$

For simplicity, we omit the subscript $\mathbf{w}'$ of g and $\mathbf{m}$ in the following when it is clear from the context. We also let $\mathcal{M}$ denote the index set of components that remain in the parameter vector (i.e., are not pruned), which corresponds to the indices of all non-zero values of the mask $\mathbf{m}(k)$.

Empirical risk reduction. Using first-order approximation and (1), we can obtain

$$F(\mathbf{w}'(k)) - F(\mathbf{w}(k+1)) \propto \sum_{j\in\mathcal{M}} g_j^2 =: \Delta(\mathcal{M}) \tag{2}$$

where "$\propto$" denotes "approximately proportional to", g_j is the j-th component of $\mathbf{g}(k)$, and we define the set function $\Delta(\mathcal{M})$ in the last equation. Details of the derivation are given in Appendix A.1. 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. Details on how to compute the gradients in further pruning during FL are given in Appendix C.5.

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 j-th parameter component. Note that this is a linear function which is sufficient according to our empirical observations, as we show in Appendix E.1. 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 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 non-linear $T(\mathcal{M})$ is given in Appendix A.2.

Optimization objective. We would like to find the optimal 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 pruned parameter vector $\mathbf{w}'(k)$, as defined in (2). 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 $\Gamma(\mathcal{M}) := \frac{\Delta(\mathcal{M})}{T(\mathcal{M})}$, we also need to ensure that $F(\mathbf{w}'(k)) \approx F(\mathbf{w}(k))$. To do so, we define an index set $\overline{\mathcal{P}}$ to denote the parameters that *cannot* 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 than *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 can be expressed as the following problem:

$$\max_{\mathcal{A} \subseteq \mathcal{P}} \quad \Gamma\left(\mathcal{A} \cup \overline{\mathcal{P}}\right) \tag{3}$$

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. The algorithm for solving (3) is given in Algorithm 1, where sorting is in non-increasing 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 $O(|\mathcal{P}| \log |\mathcal{P}|)$. The next theorem shows that the result obtained from our algorithm is a global optimal solution to (3).

Algorithm 1: Solving (3)

$\mathcal{A} \leftarrow \emptyset$;
$\mathcal{S} \leftarrow \arg\text{sort}_{j \in \mathcal{P}} \ \frac{g_j^2}{t_j}$
for $j \in \mathcal{S}$ **do**
  if $\frac{g_j^2}{t_j} \geq \Gamma\left(\mathcal{A} \cup \overline{\mathcal{P}}\right)$ **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{j\}$;
  else
    break;
return $\mathcal{A}$

Theorem 1. *We have $\Gamma\left(\mathcal{A} \cup \overline{\mathcal{P}}\right) \geq \Gamma\left(\mathcal{A}' \cup \overline{\mathcal{P}}\right)$, where $\mathcal{A}$ is given by Algorithm 1 and $\mathcal{A}'$ is any subset of $\mathcal{P}$ such that $\mathcal{A}' \neq \mathcal{A}$.*

Convergence of adaptive pruning. Adaptive pruning can both increase and decrease the model size over time. The next theorem shows that the the model parameters converge to fixed values.

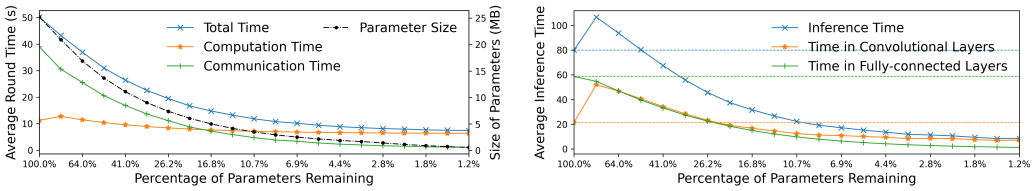
Theorem 2. *Assume that $F(\mathbf{w})$ is L-Lipschitz and β-smooth[1], and $\mathbb{E}\left[\|\mathbf{g}_{\mathbf{w}'}(k)\|^2\right] \leq \sigma^2$ for all k. When $\eta = \frac{1}{\sqrt{K}}$, the SGD recurrence in (1) for K rounds yields the following bound:*

$$\frac{1}{K}\sum_{k=0}^{K-1} \mathbb{E}[\|\nabla F(\mathbf{w}'(k)) \odot \mathbf{m}_{\mathbf{w}'}(k)\|^2 \leq \frac{C}{\sqrt{K}} + \frac{\beta\sigma^2}{2\sqrt{K}} + \frac{L}{\sqrt{K}} \sum_{k=0}^{K-1} \mathbb{E}[\|\mathbf{w}(k) - \mathbf{w}'(k)\|] \tag{4}$$

where $C := F(\mathbf{w}(0)) - F(\mathbf{w}^*)$ and $\mathbf{w}^* := \arg\min F(\mathbf{w})$.

If we do not prune in an iteration k, we have $\mathbf{w}(k) = \mathbf{w}'(k)$. 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

[1]These smoothness assumptions are commonly used in related work of convergence analysis [22, 35].



(a) Training time (b) Inference time

Figure 1: Training and Inference time on Raspberry Pi 4 devices (Conv-2 on FEMNIST).

number of *non-zero* 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. See Appendix A.3 for some further discussions.

4 Experiments

Datasets and models. We evaluate on three image datasets: (i) FEMNIST on a Conv-2 model [4], (ii) CIFAR-10 [16] on a VGG-11 model [30], and (iii) ImageNet [5] on a ResNet-18 model [10], all of which represent typical FL tasks. Note that due to practical considerations of edge devices' training time and storage capacity, we only select data corresponding to 193 writers for FEMNIST, and the first 100 classes of the ImageNet dataset (referred to as ImageNet-100 in the following).

Platform. We conduct experiments in (i) 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 (we have 10 clients in total), and (ii) a simulated setting where computation and communication times are obtained from time measurements on the prototype system involving either Raspberry Pi devices (for FEMNIST and CIFAR-10) or Android phones (for ImageNet-100). We found that our prototype and simulation results are very similar (see Appendix E.3), so we ran most experiments in simulation due to efficiency considerations. We consider FL with full client participation in the main paper and give results with random client selection [2] in Appendix E.7. The results are similar. FEMNIST data are partitioned into clients in a non-IID manner according to writer identity[2], and CIFAR-10 and ImageNet-100 are partitioned into clients in an IID manner.

Baselines. We compare the test accuracy vs. *time* curve of adaptive pruning with four baselines: (i) conventional FL [25], (ii) iterative pruning [9], (iii) online learning [8], and (iv) SNIP [19]. 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 our adaptive pruning algorithm at convergence (see Figure 4). Additional baselines are also considered in Appendix E.4.

Our main results are summarized as follows. More details on setup and additional results are given in Appendices D and E, respectively.

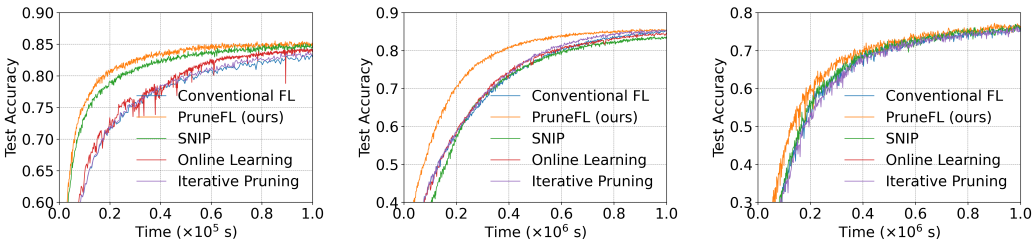
Per-round training and inference time. Figure 1 shows that our PruneFL can substantially reduce both the training and inference time, where the former is measured for one FL round and includes both communication and computation, the latter is measured for a total of 200 data samples. The computation time reduction for training is smaller than other settings since we only implement sparse computation for forward passes (see Appendix C.3).

Time to reach target accuracy. Table 1 lists the time that an algorithm first reaches a certain accuracy with FEMNIST dataset with Conv-2 model. For example, to reach 80% accuracy, PruneFL takes less than $1/3$ of time compared to conventional FL, and it also saves 27% of time (more than 100 minutes) compared to SNIP.

Table 1: Time to reach target accuracy (FEMNIST)

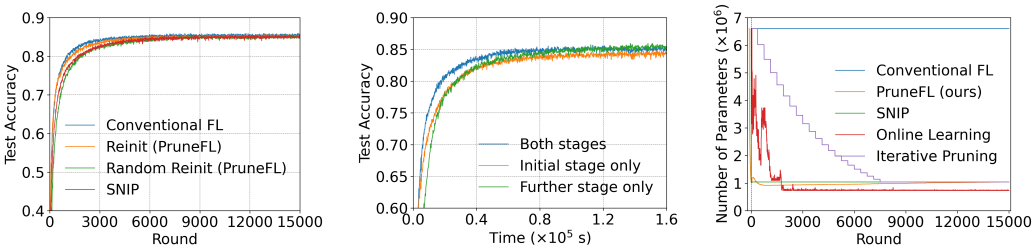
Approach	Time to reach 80% accuracy (s)	Time to reach 85% accuracy (s)
Conventional FL	51,643	188,540
PruneFL (ours)	**17,133**	**60,049**
SNIP	23,533	86,579
Online	45,328	185,059
Iterative	51,229	221,557

[2]When using full client participation, because we only have 10 clients, we partition all the 193 writers' data into 10 clients (the first 9 clients each has 19 writers' data and the last client has 22 writers' data; note that this partition is still non-IID).



(a) Conv-2 on FEMNIST (b) VGG-11 on CIFAR-10 (c) ResNet-18 on ImageNet-100

Figure 2: Test accuracy vs. time results of 3 datasets.



(a) (b) (c)

Figure 3: (a) Lottery ticket results; (b) Comparing PruneFL with approaches that only include either initial or further pruning stage; (c) Number of parameters vs. round. All use Conv-2 on FEMNIST.

Accuracy vs. training time. In Figure 2, PruneFL demonstrates a clear and consistent advantage on training speed over baselines. Moreover, PruneFL always converges to similar accuracy achieved by conventional FL. Other methods may have suboptimal performance, e.g., SNIP does not converge to conventional FL's accuracy with CIFAR-10 (Figure 2(b)), and online learning does not achieve 30% accuracy with ImageNet-100 (hence, it is below the minimum accuracy shown in Figure 2(c)).

Lottery ticket analysis. To verify whether the final model from adaptive pruning is a lottery ticket [6, 26], we reinitialize this converged model using the original random seed, and compare its accuracy vs. *round* curve with (i) conventional FL, (ii) random reinitialization (same architecture as the lottery ticket but initialized with a different random seed), and (iii) SNIP. We show in Figure 3(a) that, unlike some existing pruning techniques such as dynamic pruning [23] and SNIP [19], PruneFL finds a lottery ticket (although not necessarily the smallest).

Necessity of two-stage pruning. Figure 3(b) compares the test accuracy vs. round curves of PruneFL with methods that only include either initial pruning (at a single client) or 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 causes a slower learning speed.

Model size adaptation. An illustration of the change in model size is shown in Figure 3(c). The small negative part on x-axis shows the initial pruning stage which is unique to PruneFL (where a "round" is equivalent to 5 iterations as in our FL setting). It is worth mentioning that determining a proper target density for pruning is non-trivial. We plot two cases using SNIP to prune Conv-2 (with FEMNIST dataset) to 30% and 1% in Figure 4. It is clear that if the density is 30%, the training speed becomes slower, and if the density is 1%, the sparse model cannot converge to the optimal accuracy. In comparison, PruneFL automatically determines a proper density.

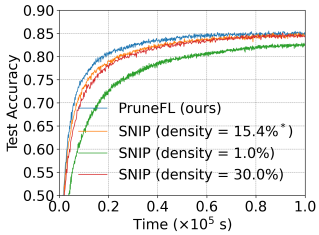


Figure 4: SNIP with different densities (FEMNIST).

5 Conclusion

We have proposed PruneFL to effectively reduce the size of neural network models so that resource-limited clients can train them within a short amount of time. Our experiments on Raspberry Pi devices confirm that we improve the cost-efficiency of FL while maintaining a similar accuracy and obtaining a lottery ticket. Our method can be applied together with other compression techniques, such as quantization, to further reduce the communication overhead.

Acknowledgment

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. V. Valls has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 795244.

References

[1] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Nikola Konstantinov, Sarit Khirirat, and Cédric Renggli. The convergence of sparsified gradient methods. In *Advances in Neural Information Processing Systems*, pages 5973–5983, 2018.

[2] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[3] Sebastian Caldas, Jakub Konečny, H Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210*, 2018.

[4] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konecný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. LEAF: A benchmark for federated settings. *CoRR*, abs/1812.01097, 2018.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[6] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] Pengchao Han, Shiqiang Wang, and Kin K Leung. Adaptive gradient sparsification for efficient federated learning: An online learning approach. *arXiv preprint arXiv:2001.04756*, 2020.

[9] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[12] Peter Kairouz, H. Brendan McMahan, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[13] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. *arXiv preprint arXiv:1910.06378*, 2019.

[14] Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian Stich, and Martin Jaggi. Error feedback fixes SignSGD and other gradient compression schemes. In *International Conference on Machine Learning*, volume 97, pages 3252–3261, Jun. 2019.

[15] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[16] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[18] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[19] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019.

[20] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *arXiv preprint arXiv:1908.07873*, 2019.

[21] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. In *International Conference on Learning Representations*, 2020.

[22] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2020.

[23] Tao Lin, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2020.

[24] Sangkug Lym, Esha Choukse, Siavash Zangeneh, Wei Wen, Sujay Sanghavi, and Mattan Erez. Prunetrain: fast neural network training by dynamic sparse model reconfiguration. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2019.

[25] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

[26] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *Advances in Neural Information Processing Systems*, pages 4933–4943, 2019.

[27] Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. Exploring sparsity in recurrent neural networks. In *International Conference on Learning Representations*, 2017.

[28] Jihong Park, Sumudu Samarakoon, Mehdi Bennis, and Mérouane Debbah. Wireless network intelligence at the edge. *Proceedings of the IEEE*, 107(11):2204–2239, 2019.

[29] Shaohuai Shi, Kaiyong Zhao, Qiang Wang, Zhenheng Tang, and Xiaowen Chu. A convergence analysis of distributed sgd with communication-efficient gradient sparsification. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 3411–3417, 2019.

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[31] Jianyu Wang and Gauri Joshi. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd. In *Machine Learning and Systems (MLSys)*, 2019.

[32] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, June 2019.

[33] Zirui Xu, Zhao Yang, Jinjun Xiong, Jianlei Yang, and Xiang Chen. Elfish: Resource-aware federated learning on heterogeneous edge devices. *arXiv preprint arXiv:1912.01684*, 2019.

[34] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):12, 2019.

[35] Hao Yu, Sen Yang, and Shenghuo Zhu. Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5693–5700, 2019.

[36] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

Appendix

A Further Details on Adaptive Pruning

A.1 Deriving Approximate Empirical Risk Reduction

We have

$$F(\mathbf{w}(k+1)) \approx F(\mathbf{w}'(k)) + \langle \nabla F(\mathbf{w}'(k)), \mathbf{w}(k+1) - \mathbf{w}'(k) \rangle \tag{5}$$
$$= F(\mathbf{w}'(k)) - \eta \langle \nabla F(\mathbf{w}'(k)), \mathbf{g}(k) \odot \mathbf{m}(k) \rangle \tag{6}$$
$$\approx F(\mathbf{w}(k)) - \eta \|\mathbf{g}(k) \odot \mathbf{m}(k)\|^2 \tag{7}$$

where $\langle \cdot, \cdot \rangle$ is the inner product, (5) is from Taylor expansion, (6) is because of (1), and (7) is obtained by using the stochastic gradient to approximate its expectation, i.e., $\mathbf{g}(k) \approx \nabla F(\mathbf{w}'(k))$. Then, the approximate decrease of empirical risk after the SGD step (1) is:

$$\begin{aligned} F(\mathbf{w}'(k)) - F(\mathbf{w}(k+1)) &\approx \eta \|\mathbf{g}(k) \odot \mathbf{m}(k)\|^2 \\ &\propto \|\mathbf{g}(k) \odot \mathbf{m}(k)\|^2 \\ &= \sum_{j \in \mathcal{M}} g_j^2 =: \Delta(\mathcal{M}). \end{aligned} \tag{8}$$

A.2 Extension to Non-linear $T(\mathcal{M})$

For the case where $T(\mathcal{M})$ is non-linear, but a general monotone and positive set function instead, we can still find a local optimal solution to (3) using Algorithm 2. We can see that the complexity of Algorithm 2 is $O(|\mathcal{P}|^2)$.

Algorithm 2: Solving (3), general $T(\cdot)$

$\mathcal{A} \leftarrow \emptyset$;
$j^* \leftarrow$ None;
repeat
  if j^* *is not None* **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{j^*\}$;
  $j^* \leftarrow \arg\max_{j \in \mathcal{P} \setminus \mathcal{A}} \frac{g_j^2}{t_j(\mathcal{A} \cup \overline{\mathcal{P}})}$;
until $\frac{g_j^2}{t_j(\mathcal{A} \cup \overline{\mathcal{P}})} < \Gamma\left(\mathcal{A} \cup \overline{\mathcal{P}}\right)$;
return $\mathcal{A}$; `// final result`

Theorem 3. *For* general $T(\mathcal{M})$*, we have* $\Gamma\left(\mathcal{A} \cup \overline{\mathcal{P}}\right) \geq \Gamma\left(\mathcal{A}' \cup \overline{\mathcal{P}}\right)$*, where* $\mathcal{A}$ *is given by Algorithm 2 and* $\mathcal{A}' = \mathcal{A} \cup \{j\}$ *for any* $j \in \mathcal{P} \setminus \mathcal{A}$*.*

Theorem 3 shows that for general $T(\mathcal{M})$, adding another component to $\mathcal{A}$ cannot improve the solution to (3).

Remark. Theorem 3 gives a weaker result for general $T(\cdot)$ compared to the global optimality result in Theorem 1 for linear $T(\cdot)$, because when $\mathcal{A}$ and $\mathcal{A}'$ differ by more than one element, it is non-straightforward to express the change in cost for general $T(\cdot)$. Furthermore, there may exist multiple local optimal solutions for general $T(\cdot)$.

A.3 Further Discussions

Convergence. This convergence result in Theorem 2 shows that the gradient components corresponding to the remaining (i.e., not pruned) parameters vanishes 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.

Note that Theorem 2 only applies to cases where (1) holds for each iteration, which corresponds to the FL setup with only one local SGD step in each round and unbiased gradient sampling across clients. Our experiments presented later show that our algorithm also converges in more general FL settings. Furthermore, 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. A theoretical analysis of these aspects is left for future work.

Tracking a small model. By choosing properly sized $\overline{\mathcal{P}}$ 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 "trainablity" in future iterations. Parameter components for which the corresponding gradient components remain zero (or close to zero) will be pruned.

In cases where there is a target maximum model size that 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.

B Proofs

B.1 Proof of Theorems 1 and 3

Recall that $\Gamma(\mathcal{M}) := \frac{\Delta(\mathcal{M})}{T(\mathcal{M})}$, where $\Delta(\mathcal{M})$ and $T(\mathcal{M})$ are both monotone and positive functions, i.e., for any $\mathcal{M} \subseteq \mathcal{M}'$, we have $0 \leq \Delta(\mathcal{M}) \leq \Delta(\mathcal{M}')$ and $0 \leq T(\mathcal{M}) \leq T(\mathcal{M}')$.

Lemma 1. *For any $\mathcal{M}$ and $\mathcal{M}'$, let $\delta_\Delta(\mathcal{M}, \mathcal{M}') := \Delta(\mathcal{M}') - \Delta(\mathcal{M})$ and $\delta_T(\mathcal{M}, \mathcal{M}') := T(\mathcal{M}') - T(\mathcal{M})$. We have $\Gamma(\mathcal{M}') \leq \Gamma(\mathcal{M})$ if and only if $\delta_\Delta(\mathcal{M}, \mathcal{M}') \leq \Gamma(\mathcal{M}) \cdot \delta_T(\mathcal{M}, \mathcal{M}')$.*

Proof.

$$
\begin{aligned}
&\Gamma(\mathcal{M}') := \frac{\Delta(\mathcal{M}')}{T(\mathcal{M}')} \leq \Gamma(\mathcal{M}) \\
&\iff \Delta(\mathcal{M}') \leq \Gamma(\mathcal{M}) \cdot T(\mathcal{M}') \\
&\iff \Delta(\mathcal{M}) + \delta_\Delta(\mathcal{M}, \mathcal{M}') \leq \Gamma(\mathcal{M}) \cdot T(\mathcal{M}) + \Gamma(\mathcal{M}) \cdot \delta_T(\mathcal{M}, \mathcal{M}') \\
&\qquad\qquad \text{(by definition of } \delta_\Delta(\cdot,\cdot) \text{ and } \delta_T(\cdot,\cdot)) \\
&\iff \Delta(\mathcal{M}) + \delta_\Delta(\mathcal{M}, \mathcal{M}') \leq \Delta(\mathcal{M}) + \Gamma(\mathcal{M}) \cdot \delta_T(\mathcal{M}, \mathcal{M}') \qquad \text{(by definition of } \Gamma(\mathcal{M})) \\
&\iff \delta_\Delta(\mathcal{M}, \mathcal{M}') \leq \Gamma(\mathcal{M}) \cdot \delta_T(\mathcal{M}, \mathcal{M}') \ .
\end{aligned}
$$

□

We are now ready to prove Theorems 1 and 3.

Proof of Theorem 1. By definition, we have $\Delta(\mathcal{M}) := \sum_{j \in \mathcal{M}} g_j^2$ and $T(\mathcal{M}) := c + \sum_{j \in \mathcal{M}} t_j$ for any $\mathcal{M}$.

In the following, we let $\mathcal{M} := \mathcal{A} \cup \overline{\mathcal{P}}$ and $\mathcal{M}' := \mathcal{A}' \cup \overline{\mathcal{P}}$. We have

$$
\begin{aligned}
\delta_\Delta(\mathcal{M}, \mathcal{M}') &= \Delta(\mathcal{M}') - \Delta(\mathcal{M}) \\
&= \sum_{j \in \mathcal{M}' \setminus \mathcal{M}} g_j^2 - \sum_{j \in \mathcal{M} \setminus \mathcal{M}'} g_j^2 \qquad (9) \\
\delta_T(\mathcal{M}, \mathcal{M}') &= T(\mathcal{M}') - T(\mathcal{M}) \\
&= \sum_{j \in \mathcal{M}' \setminus \mathcal{M}} t_j - \sum_{j \in \mathcal{M} \setminus \mathcal{M}'} t_j \ . \qquad (10)
\end{aligned}
$$

For $\mathcal{A}$ obtained from Algorithm 1, we can easily see that $\frac{g_j^2}{t_j} < \Gamma(\mathcal{M})$ for any $j \in \mathcal{M}' \setminus \mathcal{M}$ and $\frac{g_{j'}^2}{t_{j'}} \geq \Gamma(\mathcal{M})$ for any $j' \in \mathcal{M} \setminus \mathcal{M}'$. Hence,

$$\sum_{j \in \mathcal{M}' \setminus \mathcal{M}} g_j^2 < \Gamma(\mathcal{M}) \cdot \sum_{j \in \mathcal{M}' \setminus \mathcal{M}} t_j \tag{11}$$

$$\sum_{j \in \mathcal{M} \setminus \mathcal{M}'} g_j^2 \geq \Gamma(\mathcal{M}) \cdot \sum_{j \in \mathcal{M} \setminus \mathcal{M}'} t_j \ . \tag{12}$$

Combining with (9) and (10), we have

$$\delta_\Delta(\mathcal{M}, \mathcal{M}') \leq \Gamma(\mathcal{M}) \cdot \delta_T(\mathcal{M}, \mathcal{M}') \ .$$

Then, the result follows from Lemma 1. □

Proof of Theorem 3. Let $\mathcal{M} := \mathcal{A} \cup \overline{\mathcal{P}}$ and $\mathcal{M}' := \mathcal{A}' \cup \overline{\mathcal{P}}$ in this proof. As $\mathcal{A}' := \mathcal{A} \cup \{j\}$ for some $j \notin \mathcal{A}$ by definition in this theorem, we note that $\delta_\Delta(\mathcal{M}, \mathcal{M}') = g_j^2$ and $\delta_T(\mathcal{M}, \mathcal{M}') = t_j(\mathcal{M})$.

For $\mathcal{A}$ obtained from Algorithm 2, it is easy to see that $\frac{g_j^2}{t_j(\mathcal{M})} < \Gamma(\mathcal{M})$ for $j \notin \mathcal{A}$. Hence, $\delta_\Delta(\mathcal{M}, \mathcal{M}') < \Gamma(\mathcal{M}) \cdot \delta_T(\mathcal{M}, \mathcal{M}')$ and the result follows from Lemma 1. □

B.2 Proof of Theorem 2

For simplicity, we write $\mathbf{g}_{\mathbf{w}'}(k)$ and $\mathbf{m}_{\mathbf{w}'}(k)$ as $\mathbf{g}(k)$ and $\mathbf{m}(k)$ in the following. Then, (1) can be rewritten as:

$$\mathbf{w}(k+1) = \mathbf{w}'(k) - \eta \mathbf{g}(k) \odot \mathbf{m}(k) \ . \tag{13}$$

We have

$$\begin{aligned}
\mathbb{E}\left[F(\mathbf{w}(k+1)) | \mathbf{w}'(k), \mathbf{m}(k)\right] &\leq F(\mathbf{w}'(k)) - \eta \left\langle \nabla F(\mathbf{w}'(k)), \mathbb{E}\left[\mathbf{g}(k) \odot \mathbf{m}(k) | \mathbf{w}'(k), \mathbf{m}(k)\right] \right\rangle \\
&\quad + \frac{\eta^2 \beta}{2} \mathbb{E}\left[\left\| \mathbf{g}(k) \odot \mathbf{m}(k) \right\|^2 \middle| \mathbf{w}'(k), \mathbf{m}(k) \right] \qquad (14) \\
&= F(\mathbf{w}'(k)) - \eta \left\| \nabla F(\mathbf{w}'(k)) \odot \mathbf{m}(k) \right\|^2 \\
&\quad + \frac{\eta^2 \beta}{2} \mathbb{E}\left[\left\| \mathbf{g}(k) \odot \mathbf{m}(k) \right\|^2 \middle| \mathbf{w}'(k), \mathbf{m}(k) \right] \qquad (15) \\
&\leq F(\mathbf{w}(k)) + L \left\| \mathbf{w}(k) - \mathbf{w}'(k) \right\| - \eta \left\| \nabla F(\mathbf{w}'(k)) \odot \mathbf{m}(k) \right\|^2 \\
&\quad + \frac{\eta^2 \beta}{2} \mathbb{E}\left[\left\| \mathbf{g}(k) \odot \mathbf{m}(k) \right\|^2 \middle| \mathbf{w}'(k), \mathbf{m}(k) \right] \qquad (16)
\end{aligned}$$

where each step is explained as follows. Equation (14) is due to β-smoothness and the update equation (13). Equation (15) is obtained from

$$\begin{aligned}
\mathbb{E}\left[\mathbf{g}(k) \odot \mathbf{m}(k) | \mathbf{w}'(k), \mathbf{m}(k)\right] &= \mathbb{E}\left[\mathbf{g}(k) | \mathbf{w}'(k)\right] \odot \mathbf{m}(k) \\
&= \nabla F(\mathbf{w}'(k)) \odot \mathbf{m}(k)
\end{aligned}$$

and

$$\begin{aligned}
&\left\langle \nabla F(\mathbf{w}'(k)), \nabla F(\mathbf{w}'(k)) \odot \mathbf{m}(k) \right\rangle \\
&\quad = \left\langle \nabla F(\mathbf{w}'(k)) \odot \mathbf{m}(k), \nabla F(\mathbf{w}'(k)) \odot \mathbf{m}(k) \right\rangle \\
&\quad = \left\| \nabla F(\mathbf{w}'(k)) \odot \mathbf{m}(k) \right\|^2
\end{aligned}$$

because multiplying $\mathbf{m}(k)$ to the first part of the inner product does not change the result as the same mask $\mathbf{m}(k)$ is multiplied to the second part of the inner product and $\mathbf{m}(k)$ has components of either zero or one. Equation (16) is due to L-Lipschitzness such that $F(\mathbf{w}'(k)) - F(\mathbf{w}(k)) \leq L \left\| \mathbf{w}(k) - \mathbf{w}'(k) \right\|$.

The conditional expectation above is conditioned on both $\mathbf{w}'(k)$ and $\mathbf{m}(k)$, because a zero component in $\mathbf{w}'(k)$ does not necessarily imply a zero component in the mask $\mathbf{m}(k)$, and the mask $\mathbf{m}(k)$ does not convey information about the actual values in $\mathbf{w}'(k)$.

Taking expectation on both sides of (16), we have

$$\begin{aligned}&\mathbb{E}\left[F(\mathbf{w}(k+1))\right]\\&\quad\leq \mathbb{E}\left[F(\mathbf{w}(k))\right]+L\mathbb{E}\left[\|\mathbf{w}(k)-\mathbf{w}'(k)\|\right]-\eta\mathbb{E}\left[\|\nabla F(\mathbf{w}'(k))\odot\mathbf{m}(k)\|^2\right]+\frac{\eta^2\beta\sigma^2}{2}\end{aligned}\tag{17}$$

where the last term is because $\mathbb{E}\left[\|\mathbf{g}(k)\odot\mathbf{m}(k)\|^2\right]\leq\mathbb{E}\left[\|\mathbf{g}(k)\|^2\right]\leq\sigma^2$ as $\mathbf{m}(k)$ has components of zero or one.

By rearranging, we obtain

$$\begin{aligned}&\mathbb{E}\left[\|\nabla F(\mathbf{w}'(k))\odot\mathbf{m}(k)\|^2\right]\\&\quad\leq\frac{\mathbb{E}\left[F(\mathbf{w}(k))\right]-\mathbb{E}\left[F(\mathbf{w}(k+1))\right]}{\eta}+\frac{L}{\eta}\mathbb{E}\left[\|\mathbf{w}(k)-\mathbf{w}'(k)\|\right]+\frac{\eta\beta\sigma^2}{2}\ .\end{aligned}\tag{18}$$

Hence,

$$\begin{aligned}&\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\left[\|\nabla F(\mathbf{w}'(k))\odot\mathbf{m}(k)\|^2\right]\\&\quad\leq\frac{\mathbb{E}\left[F(\mathbf{w}(0))\right]-\mathbb{E}\left[F(\mathbf{w}(K))\right]}{\eta K}+\frac{L}{\eta K}\sum_{k=0}^{K-1}\mathbb{E}\left[\|\mathbf{w}(k)-\mathbf{w}'(k)\|\right]+\frac{\eta\beta\sigma^2}{2}\end{aligned}\tag{19}$$

$$\leq\frac{F(\mathbf{w}(0))-F(\mathbf{w}^*)}{\eta K}+\frac{L}{\eta K}\sum_{k=0}^{K-1}\mathbb{E}\left[\|\mathbf{w}(k)-\mathbf{w}'(k)\|\right]+\frac{\eta\beta\sigma^2}{2}\ .\tag{20}$$

The result is proven by letting $\eta=\frac{1}{\sqrt{K}}$ and rearranging the last two terms in (20). □

C Implementation Details

C.1 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 [9], 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-size models, and sparse matrices for weights in both convolutional and fully-connected layers in pruned models.

C.2 Complexity Analysis

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 non-zero 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 $\min\left\{2\times d,\frac{1}{32}+d\right\}$, where d is the model's *density* (percentage of non-zero parameters). This ratio is further optimized when the matrix sparsity pattern is fixed (in most FL rounds, see Appendix C.5). In this case, there is no extra cost since only values of the non-zero entries need to be exchanged.

Computation. Because dense matrix multiplication is extremely optimized, sparse matrices will show 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 non-zero entries in $\mathbf{S}$ (assuming $\mathbf{D}$ is fixed).

C.3 Implementation Challenges

As of today, well-known machine learning frameworks have limited support of 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; and sparse matrices are not supported for the kernels in convolutional layers, etc. 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 in forward passes. We do not improve backward passes due to implementation limitations (more details in Appendix C.4). 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.

C.4 Gradient Computation

The forward pass in neural networks with sparse matrices is straightforward: the input data is multiplied by a sparse weight, and produces a dense output to be passed to the next layer. The backward pass, however, is different. Taking an FC layer as an example (convolutional layers are more complex but similar in principle), let $\mathbf{u}$ be its weight, and we assume there is no bias. Then, the gradient of $\mathbf{u}$ is given by

$$\mathbf{g}_{\mathbf{u}} = \mathbf{x}^{\mathrm{T}} \mathbf{g}_{\text{out}}.$$

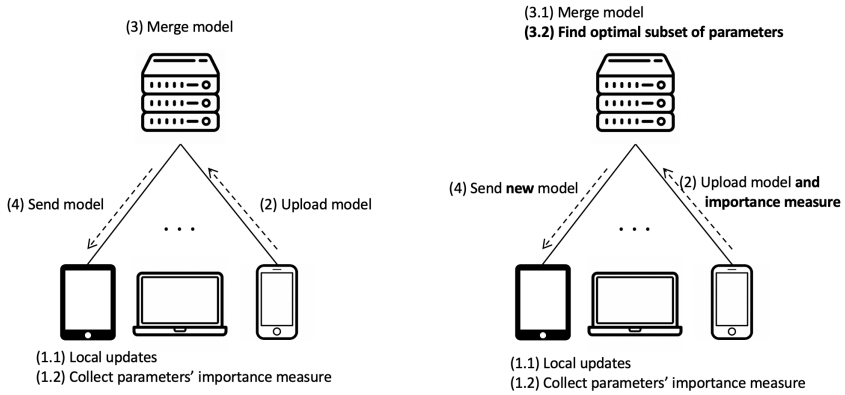
Here, $\mathbf{x}$ is the (dense) input of size $N \times n_{\text{in}}$, and $\mathbf{g}_{\text{out}}$ is the (dense) gradient in backpropagation fed by the next layer (size is $N \times n_{\text{out}}$), where n_{in}, n_{out}, and N are the number of input neurons, output neurons of the FC layer, and the batch size for SGD, respectively. The weight $\mathbf{u}$ is of size $n_{\text{in}} \times n_{\text{out}}$. Note that both $\mathbf{x}$ and $\mathbf{g}_{\text{out}}$ are dense, and thus current implementations (e.g., PyTorch) first compute the dense gradient with $\mathbf{u}$'s dense form that has all zero values included, and then select values from the dense gradient according to $\mathbf{u}$'s sparse pattern. There is currently no better way to accelerate this process as far as we know. Therefore, this implementation does not improve the backward pass's speed. It improves the forward pass only.

For the above reason, in our implementation we collect the gradients of zero-valued components of $\mathbf{u}$ at the same time with no extra overhead (although those zero-valued components themselves are not updated). This characteristic is useful in our adaptive pruning procedure.

C.5 Information Exchange During Further Pruning

During the further pruning stage, the stochastic gradient $\mathbf{g}(k)$ 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 $\mathbf{g}(k)$ 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).

This section provides detailed information for the adaptive pruning procedure described in Section 3.2. There are two cases of FL round in adaptive pruning: non-reconfiguration round and reconfiguration round. In every local update within a non-reconfiguration round, each client collects $g_i^2 \ \forall i$, the squared gradient for all (including pruned and remaining) parameters in the model. This is done by performing an extra inner product on the gradient vector after each of the client's local update. This set of squared gradients is summed locally after every local update, until the sum is sent to the server in the next reconfiguration round. Note that in non-reconfiguration rounds, only remaining model parameters are used in computation and exchanged between server and clients. Since the parameter



(a) Non-reconfiguration round (b) Reconfiguration round

Figure 5: Illustration of adaptive pruning as part of further pruning during FL.

set is fixed in non-reconfiguration rounds, only the *values* of the parameters need to be exchanged between the server and clients, which incurs no extra communication cost.

Reconfiguration rounds happen on the server periodically between non-reconfiguration rounds. Clients that have participated in FL at least once since the last reconfiguration round are notified by the server. In a reconfiguration round, each client that has participated since the previous reconfiguration round sends the "importance measure", i.e., summed squared gradients of each parameter to the server. The server can obtain $\overline{g_i^2}\ \forall i$, the average of each parameter's squared gradient (possibly weighted by the number of processed data associated with each client). The server then uses the gradient information, along with an estimated time cost associated with each parameter to find an optimal subnetwork that maximizes the decrease in empirical risk per unit time. Finally, the server sends the new subnetwork back to clients; clients remove the previously collected importance measure and run non-reconfiguration rounds on the new subnetwork until next reconfiguration. The reader can find an illustration of the two types of rounds in Figure 5.

C.6 PyTorch on Raspberry Pi devices

To install PyTorch on Raspberry Pi devices, we follow the instructions described at `https://bit.ly/3e6I7tG`, where acceleration packages such as MLKDNN and NNPACK are disabled due to possible compatibility issues and their lack of support of sparse computation. We compare our implementation with the *plain* implementation by PyTorch without accelerations. We expect that similar results can be obtained if acceleration packages could support sparse computation. This is an active area of research on its own where methods for efficient sparse computation on both CPU and GPU have been developed in recent years. Integrating such methods into our experiments is left for future work.

D Experiment Setup Details

Dataset details. FEMNIST dataset contains 28×28 images from 62 classes of handwritten digits and letters (including lower and upper cases). It is collected from 3,500 different writers. To fit into the storage space of Raspberry Pi devices, we use a subset from 193 writers which has 35,948 training data samples and 4,090 test data samples. CIFAR-10 consists of 50,000 training and 10,000 test images (32×32 size) with three-channelled color. ImageNet contains various sizes of images, considering the capacity of edge devices, we use the first 100 classes of it (called ImageNet-100), which contains 126,100 training images. We use the validation set of 5,000 images to evaluate our algorithm on ImageNet-100. We list the detailed configurations in Table 2.

Model architecture details. The architecture details are presented in Table 3. VGG-11 and ResNet-18 are well-known architectures, and we directly acquire Conv-2 from its original work [4]. We

Table 2: Evaluation configurations (C.S. stands for client selection; LR stands for learning rate).

Dataset	FEMNIST	CIFAR-10	ImageNet-100
SGD params in round r	LR = 0.25	LR = $0.1 \times 0.5^{\frac{r}{10000}}$	LR = $0.1 \times 0.5^{\lfloor \frac{r}{1000} \rfloor \cdot 0.1}$ momentum = 0.9
Fraction of non-zero prunable parameters in round r	$0.3 \times 0.5^{\frac{r}{10000}}$	$0.3 \times 0.5^{\frac{r}{10000}}$	$0.3 \times 0.5^{\frac{r}{10000}}$
Number of data samples used in initial pruning	200	200	500
Number of clients (Non-C.S., C.S.)	10, 193	10, 50	10, 50
Mini-batch size, local iterations each round	20, 5	20, 5	20, 5
Reconfiguration	every 50 rounds	every 50 rounds	every 50 rounds
Total number of FL rounds	15,000	15,000	15,000
Evaluation	prototype (Pi 4), simulation (Pi 4)	simulation (Pi 4)	simulation (Android VM)

Table 3: Model architectures.

Architecture	Conv-2	VGG-11	ResNet-18
Convolutional	32, pool 64, pool	64, pool, 128, pool, 2×256, pool, 2×512, pool, 2×512, pool	64, pool, $2 \times [64, 64]$ $2 \times [128, 128]$, $2 \times [256, 256]$ $2 \times [512, 512]$
Fully-connected	2048, 62 (input size: 3136)	512, 512, 10 (input size: 512)	avgpool, 100 (input size: 512)
Conv/FC/all params	52.1K/6.6M/6.6M	9.2M/530.4K/9.8M	11.2M/102.6K/11.3M

adapted some layers in VGG-11 and ResNet-18 to match with the number of output labels in our datasets.

Baseline details. Since our experiments try to minimize the training time using pruning, there is no direct way of comparison with the 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 adaptive pruning, and keep the architecture afterwards. 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 adaptive pruning, and the pruning rate is equal across layers. See Figure 3(c) for the illustration of the baseline settings.

Platform details. Unless otherwise specified, the prototype system includes nine Raspberry Pi (version 4, with 2 GB RAM, 32 GB SD card) devices as clients and a personal computer without GPU as both a client and the server (totaling 10 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 by measurements from Android virtual machine (VM).

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.

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 5 consecutive reconfigurations.

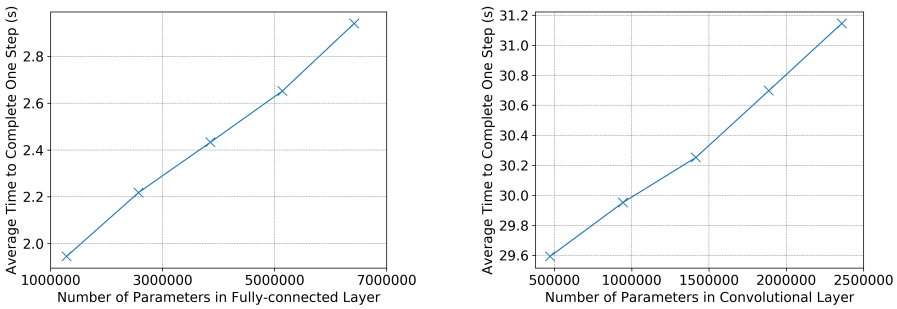
For adaptive pruning, to ensure convergence of the last term on the RHS of (4) in Theorem 2, we exponentially decrease the number of *non-zero* prunable parameters in $\mathcal{P}$ over rounds (see Table 2). We note that $\mathcal{P}$ includes both zero and non-zero parameters, hence the size of $\mathcal{P}$ 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 belong to $\overline{\mathcal{P}}$ that cannot be pruned.

Biases (if any) in the DNN are not pruned, and 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.

E Additional Experiment Results

E.1 Validation of Assumptions

Agreeing with our assumptions and analysis in Sections 3.2 and C.2, we observe that the training time in each layer is generally independent of the other layers. Within each layer, the time is approximately linear with the number of parameters in the layer with sparse implementation. In Figures 6, we fix the parameters in other layers and increase the number of parameters in Conv-2's largest (first) FC layer and VGG-11's largest (last convolutional layer with 512 channels) convolutional layer, respectively, and measure for 50 times. The R^2 values of linear regression for Conv-2 and VGG-11 are 0.997 and 0.994, respectively.



(a) Conv-2's first fully-connected layer (b) VGG-11's last convolutional layer

Figure 6: Linearity of average computation time vs. number of parameters.

E.2 Further Discussion on Computation and Communication Time Reduction

In Figure 1, 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-size Conv-2 model in dense form (100% on x-axis) as well as the pruned models in sparse form at different densities (elsewhere on x-axis), and measure the average elapsed time of FL on these pruned models involving both the server and clients over 10 rounds.

Computation time. We see from Figure 1(a) that as the model density decreases, the computation time (for five local iterations) decreases from 11.24 seconds per round to 6.34 seconds per round. This reduction in computation time is moderate since we only implement sparse computation for forward passes (see Section C.3). Additionally, we plot in Figure 1(b) the total inference time for 200 data samples, which shows the similar trends as in Figure 1(a).

Communication time. Our implementation of sparse matrices reduces the storage requirement significantly (see Appendix C). Compared with computation time, the decrease in the communication time is more noticeable. It drops from 35.88 seconds per round to 1.04 seconds per round.

Enabling FL on low-power edge device. In addition, we observe that when training the MNIST [17] dataset on the full-size, dense-form LeNet-300-100 model [17] on Raspberry Pi *version 3* (with 1 GB RAM, 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 impossible on Raspberry Pi 3.

E.3 Comparing Conventional FL and PruneFL

Figure 7 shows the test accuracy vs. 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 seconds) 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 because training CIFAR-10 and ImageNet-100 models on the prototype system take an excessive amount of time.

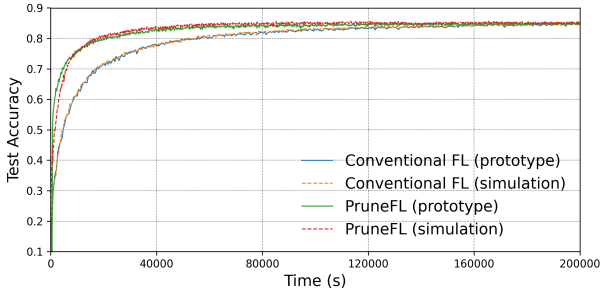


Figure 7: Comparing conventional FL and PruneFL with both prototype and simulation results (Conv-2 on FEMNIST).

E.4 Comparing Training Time with Additional Baselines

To avoid bottlenecks, our algorithm and implementation ensures that all components in PruneFL, including communication, computation, and reconfiguration, are orchestrated and inexpensive. For this reason, some approaches in the literature that are 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 [24] applies regularization on every input 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 seconds per round, which is a 57% increase compared to PruneFL.

Considering communication time, dynamic pruning with feedback (DPF) [23] maintains a full-sized model, and clients have to upload full-size 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 round (every 50 rounds in our experiments), and always exchange pruned models otherwise. This gives an average cost of $\frac{(1+d)+2\times 49d}{50} = 0.02 + 1.96d$ including both uploading and downloading. For instance, when the model density is 10%, DPF incurs $5.1\times$ communication cost compared to PruneFL. Finally, our reconfiguration algorithm (Algorithm 1) runs in quasi-linear time, making it possible to be implemented on edge devices.

E.5 Relative Importance Between Layers

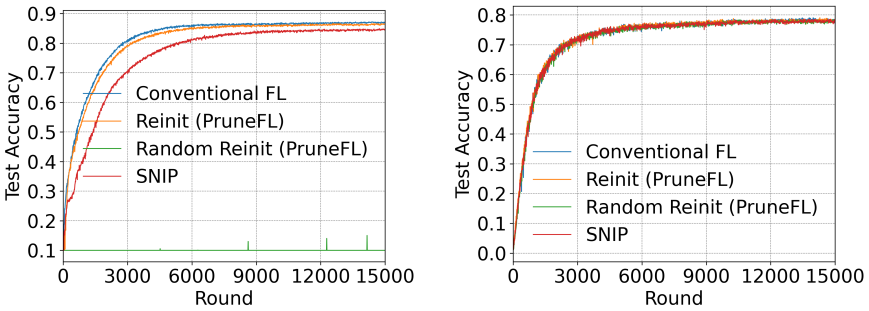
Similar to [19], our algorithm discovers the relative importance between layers automatically. The remaining densities of convolutional layers and FC layers at convergence are listed in Table 4 in a sequential manner. We observe that usually the input and output layers are not pruned to a low density, indicating that they are relatively important in the neural network architectures. This also agrees with the pruning scheme in [6], 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 remaining densities.

Table 4: Remaining density in each layer at convergence (no client selection).

	Conv-2	VGG-11	ResNet-18
Convolutional layers	0.99, 0.54	1.0, 0.89, 0.91, 0.90, 0.65, 0.13, 0.02, 0.03	0.94, 0.89, 0.75, 0.69, 0.88, 0.81, 0.92, 0.92, 0.99, 0.43, 0.98, 0.99, 1.0, 0.89, 0.69, 0.87, 0.83
Fully-connected layers	0.15, 0.38	0.10, 0.13, 0.54	0.96

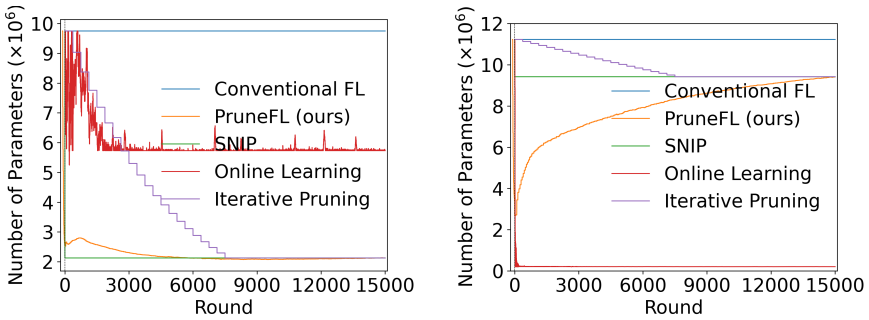
E.6 Complete Lottery Ticket and Model Size Adaptation Results (no client selection)

This section presents the lottery ticket results and model size adaptation results for the rest 2 datasets, CIFAR-10 and ImageNet-100. (Figure 8 corresponds to Figure 3(a), Figure 9 corresponds to Figure 3(c)).



(a) VGG-11 on CIFAR-10 (b) ResNet-18 on ImageNet-100

Figure 8: Lottery ticket results of VGG-11 on CIFAR-10 and ResNet-18 on ImageNet-100 (no client selection).

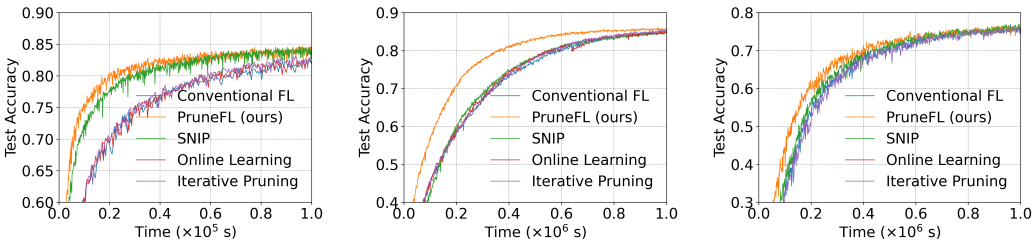


(a) VGG-11 on CIFAR-10 (b) ResNet-18 on ImageNet-100

Figure 9: Number of parameters vs. round for VGG-11 on CIFAR-10 and ResNet-18 on ImageNet-100 (client selection).

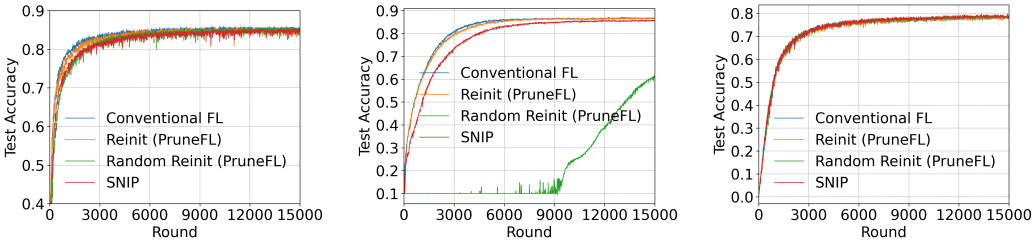
E.7 Client Selection Results for Section 4

In this section, we present simulation results under same settings as in Section 4, but with client selection. FEMNIST is collected from different writers, thus the dataset is intrinsically non-IID. Since use a subset of data from 193 writers, if client selection is used, we randomly select images from 10 out of 193 writers in each round. We partition CIFAR-10 and ImageNet-100 uniformly into 50 non-overlapping subset, respectively, and randomly select 10 out of 50 for training in each round. Figure 10 (corresponding to Figure 2) shows the training time reduction; Figure 11 (corresponding to Figure 3(a) and Figure 8) shows the lottery ticket result; and Figure 12 (corresponding to Figure 3(c) and Figure 9) shows the model size adaptation. We observe similar behaviors from the client selection results, and thus, we skip the analysis in this section.



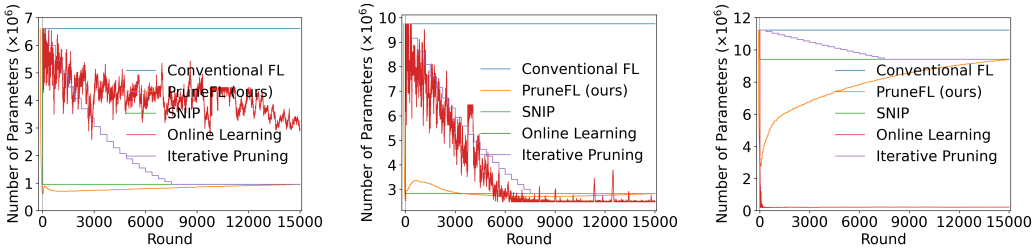
(a) Conv-2 on FEMNIST (b) VGG-11 on CIFAR-10 (c) ResNet-18 on ImageNet-100

Figure 10: Test accuracy vs. time results of 3 datasets (client selection)



(a) Conv-2 on FEMNIST (b) VGG-11 on CIFAR-10 (c) ResNet-18 on ImageNet-100

Figure 11: Lottery ticket results of 3 datasets (client selection).



(a) Conv-2 on FEMNIST (b) VGG-11 on CIFAR-10 (c) ResNet-18 on ImageNet-100

Figure 12: Number of parameters vs. round for 3 datasets (client selection).

E.8 Convergence Accuracy Results for All Experiments

The convergence accuracies with/without client selection are shown in Table 5. The results are taken from the test accuracies in the last five evaluations of each simulation. Conventional FL sometimes shows a slight advantage because all methods run for the same number of *rounds*, and full-size models in conventional FL learn faster when the accuracy is measured in *rounds* instead of time.

Table 5: Average of the last 5 measured accuracies (%). C.S. stands for client selection.

	FEMNIST		CIFAR-10		ImageNet-100	
	No C.S.	C.S.	No C.S.	C.S.	No C.S.	C.S.
Conventional FL	85.49 ± 0.21	85.06 ± 0.29	86.98 ± 0.10	86.69 ± 0.10	77.88 ± 0.62	78.15 ± 0.29
PruneFL (ours)	84.82 ± 0.34	84.72 ± 0.54	86.02 ± 0.12	86.14 ± 0.07	78.07 ± 0.57	77.54 ± 0.10
SNIP	85.02 ± 0.38	84.63 ± 0.65	84.57 ± 0.12	85.59 ± 0.10	78.05 ± 0.35	78.62 ± 0.40
Online learning	84.50 ± 0.27	84.88 ± 0.23	85.82 ± 0.08	85.47 ± 0.11	18.43 ± 1.59	25.31 ± 0.67
Iterative pruning	84.98 ± 0.11	84.35 ± 0.95	86.00 ± 0.12	85.95 ± 0.16	77.65 ± 0.31	78.17 ± 0.29

E.9 Training with Limited and 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.

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 1 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 d_l is the density limit, $d_t \leq d_l$ is the target model density at the end of the further pruning stage, and PruneFL is run for r_{max} rounds after the initial pruning stage, then the maximum density at round r is $d_{\text{max}}(r) = \frac{1}{r_{\text{max}}}(r \cdot d_t + (r_{\text{max}} - r) \cdot d_l)$. We select $d_l = 15\%$ and $d_t = 5\%$ for Conv-2 on FEMNIST. The results are given in Figure 13. We see that if we do not impose these model size constraints, PruneFL exceeds the density limits $d_l = 15\%$ 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 $d_l = 15\%$ and reaches the target density $d_t = 5\%$ at the end of training.

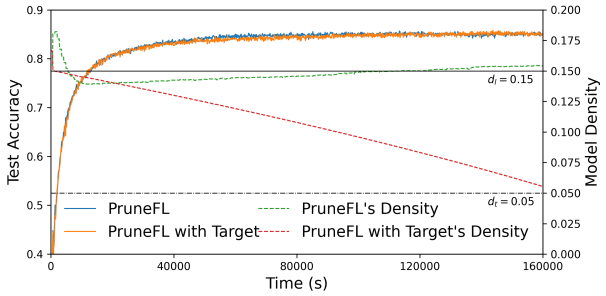


Figure 13: Conv-2 on FEMNIST with effective size limit and target.