Device-Wise Federated Network Pruning

Shangqian Gao*[1], Junyi Li*[2], Zeyu Zhang*[3], Yanfu Zhang[4], Weidong Cai[5], Heng Huang[2]†
[1] Electrical and Computer Engineering, University of Pittsburgh,
[2] Computer Science, University of Maryland College Park,
[3] Information, University of Arizona, [4] Computer Science, College of William and Mary,
[5] School of Computer Science, University of Sydney

Abstract

Neural network pruning, particularly channel pruning, is a widely used technique for compressing deep learning models to enable their deployment on edge devices with limited resources. Typically, redundant weights or structures are removed to achieve the target resource budget. Although data-driven pruning approaches have proven to be more effective, they cannot be directly applied to federated learning (FL), which has emerged as a popular technique in edge computing applications, because of distributed and confidential datasets. In response to this challenge, we design a new network pruning method for FL. We propose device-wise sub-networks for each device, assuming that the data distribution is similar within each device. These sub-networks are generated through sub-network embeddings and a hypernetwork. To further minimize memory usage and communication costs, we permanently prune the full model to remove weights that are not useful for all devices. During the FL process, we simultaneously train the device-wise sub-networks and the base sub-network to facilitate the pruning process. We then finetune the pruned model with device-wise sub-networks to regain performance. Moreover, we provided the theoretical guarantee of convergence for our method. Our method achieves better performance and resource trade-off than other well-established network pruning baselines, as demonstrated through extensive experiments on CIFAR-10, CIFAR-100, and TinyImageNet.

1. Introduction

Machine learning algorithms often rely on large amounts of data, but privacy restrictions can prevent data from being easily shared across different organizations. For instance, hospitals may have isolated data that are limited in size and cannot be used to train a high-quality model with good predictive accuracy. Collaboration between organizations to train a machine learning model on their combined data can lead to better results, but sharing data is often not possible due to privacy policies and regulations [1]. This problem of 'data islands' is not limited to hospitals and can be found in other areas such as finance, government, and supply chains. Federated learning [42, 49, 74] has emerged as a popular research topic in the machine learning and computer vision communities as a solution to these issues.

Convolution Neural Networks (CNNs) have achieved remarkable success in various computer vision tasks[35, 56, 61], but to address real-world challenges, recent CNNs have become wider and deeper, leading to improved performance on various benchmarks. However, this increased capacity comes at the cost of higher computational and storage requirements, which prohibit CNNs from being deployed on edge devices. Consequently, numerous efforts [17, 55] have been made to reduce the size of CNNs to enable their deployment on mobile and embedded devices. Among different directions, weight pruning [18] and structural pruning [38] are two major ways to reduce the model size. Network pruning methods have achieved promising results. However, most existing methods do not consider heterogeneous (non-iid) local data distributions. Instead, they upload local data to the server to train and prune the model based on the whole dataset.

There are several existing works [21, 27, 37, 52, 60] on network pruning under non-iid local data distributions. These methods mainly focus on weight pruning. SCBFwP [60] tries to perform channel pruning under non-iid data distributions, but they mostly rely on channel norms as the importance score, and they did not show how to scale their method to larger CNNs. Our method is designed to perform channel pruning given a certain computational budget (measured in FLOPs) on each device. Because, unlike weight pruning, channel pruning can achieve acceleration and compression without any post-processing steps.

Previous research [22, 76] show that data-driven pruning

†This work was partially supported by NSF IIS 2347592, 2347604, 2348159, 2348169, DBI 2405416, CCF 2348306, CNS 2347617.

*These authors contributed equally to this work.

approaches often perform much better than using a fixed criterion (like channel norm [38]). Inspired by this result, we propose to discover the proper sub-network following the guidance of local data distributions. Specifically, we divide the learning of sub-networks into two parts. In the first part, we design a server-side sub-network, which is used to serve as a base model for device-wise sub-network. Ideally, weights that are not useful for any device will be removed from the server-side sub-network. In the second part, device-side sub-networks will be adaptively generated for each device. The device-side sub-network will be pruned to meet the specific resource requirement for each device. Both the server-side sub-network and device-side sub-networks are generated by mapping the device id(s) into the network embedding space. The embedding for each sub-network will then be fed into a hypernetwork [16] to generate the corresponding structure. The hypernetwork and the embedding are trained together through gradient-based optimization algorithms in a federated fashion. To control the communication costs brought by training the hypernetwork, we set the fraction of update steps for the hypernetwork to be small. So that the overhead of training these sub-networks is not large. In addition, to improve the training efficiency given the limited update budget, we perform iterative training of model weights and the hypernetwork, which makes the hypernetwork adapt to changes in model weights. The training process of our method may pose challenges to the convergence of the model. We show that our method can be converted into a bi-level optimization problem under the FL setting. We further provide the theoretical convergence guarantee showing that our method can converge to a stationary point. The contribution of this work can be summarized as follows:

- We proposed a novel channel pruning method for federated learning. A server-side network and device-wise sub-networks are learned to achieve a better trade-off between the performance and the computational resource.
- We proposed to use an embedding layer and a hypernetwork to generate sub-networks on each device. As a result, no sub-network structure needs to be stored for each device.
- We provided the theoretical guarantee of convergence for our method for federated learning by reformulating our method as a bi-level optimization problem.
- Extensive experiments on CIFAR-10, CIFAR-100, and TinyImageNet show the effectiveness of our method across different models like ResNet-56, ResNet-18/34, and MobileNet-V2.

2. Related Works

Federated Learning. Federated learning (FL) is a new kind of distributed learning approach that involves a server coordinating a group of clients/devices to learn a model. In FL, at each epoch, devices retrieve the model from the server, train the model locally for several steps, and then upload the updated model back to the server. The server aggregates the updates from devices to update the global model. FL presents several challenges that need to be addressed for effective implementation. Firstly, devices in FL often have different data distributions. Various methods are proposed to solve the data heterogeneity [20, 29, 43, 44, 51, 58, 82]. Second, the communication between devices and the server is expensive and is a critical bottleneck in FL training. Compression techniques are applied in FL to reduce the communication [24, 30, 45, 57, 64, 68], Finally, although in FL, the server does not have access to the user data, model inversion attack [15] is shown to recover the user information based on the model updates. Cryptography techniques are applied to improve the privacy of FL, such as homomorphic encryption [39, 53], differential privacy [49] and multiparty secure computation [66] *etc.*. In addition to the challenges discussed, there are several other challenges in FL, such as fairness and model interpretability, a more comprehensive review of FL can be found in [28, 42].

Network Pruning. (1) Regular Setting. Most network pruning methods assume they can easily access all samples without restrictions. Early works [18, 38] simply use L_1 or L_2 norm to measure the importance of weights or structures. Calculating norms does not require samples, and it can be seamlessly extended to the FL setting. However, the performance of these methods is often worse than methods [11, 22, 76] that require samples for pruning. One direction of data-driven pruning methods relies on batch normalization (BN) [23] layers since BN is popular for the design of recent CNNs [19, 59]. These methods utilize the scaling factor of BN to indicate which channels are important. Liu *et al.* [46] use sparse regularization on the scaling factors of BN to prune channels, where a channel is pruned if its corresponding scaling factor is small. Pruning methods that involve BN are effective. However, BN layers can pose challenges in the FL setting due to the varying data distribution across devices, leading to significant differences in BN layers' running mean and variance. Another research direction frames channel pruning as a constrained optimization problem [8–14, 32, 76]. In this direction, learnable parameters are utilized to control each channel, and these parameters are end-to-end differentiable, allowing for gradient-based optimization methods. Since these methods do not rely on BN layers, they can potentially be extended to the FL setting. Alongside advancements in vision, Natural Language Processing (NLP) has significantly progressed, demonstrated by key studies [62, 72, 77–81]. Concurrently, structure pruning has emerged as a method to improve the efficiency of large language models, as shown in recent research [67].

(2) Non-iid Setting. Without specialized treatment, regular pruning methods can impose strong biases in pruned mod-

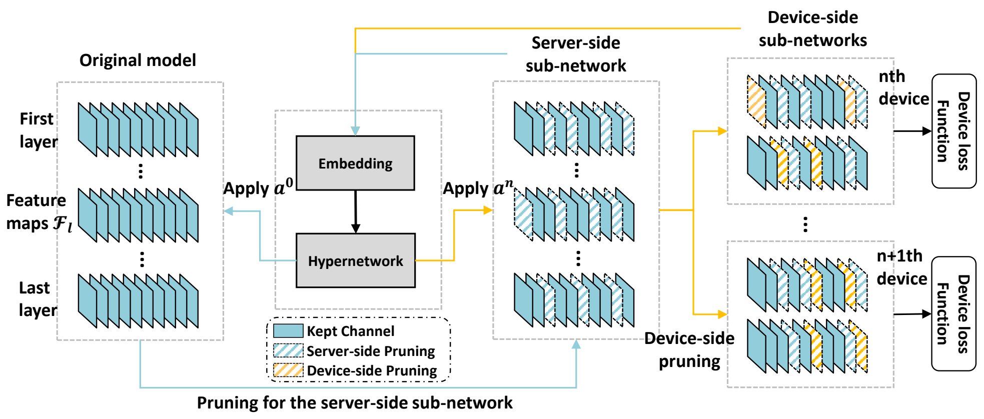


Figure 1. An overview of our proposed method. We first generate network embeddings for server-side and device-side sub-networks given their ids. We then use them as the inputs to the hypernetwork to produce the corresponding sub-networks. We then optimize the hypernetwork given loss functions on each device.

els because of heterogeneous (non-iid) local data distributions. Shao *et al.* [60] employs local training with a full-size model to discard unimportant channels (measured in channel norms) on devices. FedPrune [52] guides pruning based on updated activations. LotteryFL [37] iteratively prunes a full-size model on devices. PruneFL [27] reduces local computational costs by finer pruning a coarse-pruned model. ZeroFL [54] partitions weights into active and non-active weights and stores sparsified weights and activations for backward propagation, and it also needs to store non-active weights and dense gradients. Bibikar *et al.* [3] employs mask adjustment on devices and sparse aggregation and magnitude pruning on the server to generate a new global model. The FedTiny [21] approach incorporates an adaptive batch normalization (BN) selection module, which adaptively obtains an initially pruned model that can better fit deployment scenarios. Most aforementioned methods focus on weight-level pruning/sparsity, often requiring high communication costs to compute importance scores for all parameters. On the other hand, our method learns the channel configuration of each layer, which is less resource-demanding.

Federated Bilevel Optimization. Our channel pruning can be viewed as a federated bilevel optimization problem (FedBiO). The general FedBiO problem has been studied in the literature [65, 71, 75]. FedNest [65] studied the general nested federated problems with FedBiO being a special case, and it utilized variance reduction to tackle the heterogeneity of lower level problems; simFBO [71] and FedBiOAcc [41] adapts the single loop bilevel optimization problems to the federated learning setting. Some applications in FL can be viewed as bilevel optimization problems, such as noisy labels [40] and communication-efficient FL [41].

3. Method

3.1. Notations

We will first introduce our notations before formally describing our method. In a convolutional neural network (CNN), the feature map of the lth layer is denoted by $\mathcal{F}_l \in \Re^{C_l \times W_l \times H_l}$, where C_l represents the number of channels, and H_l and W_l represent the height and width of the current feature map. L denotes the total number of layers in the CNN. For simplicity, we ignore the mini-batch dimension of feature maps in our notation.

3.2. Federated Learning Setting

We first describe the federated learning problem considered in this paper. In the FL setting, we train a neural network on N local datasets $D_n,\ n \in \{1, 2, 3 \cdots, N\}$. Through this paper, the data distribution on local devices is heterogeneous. To train a neural network in this setting, we want to optimize the following optimization problem:

$$\min_{\mathcal{W}} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathcal{W}, D_n), \tag{1}$$

where $\mathcal{W}$ is the weights of the CNN, and $\mathcal{L}$ is the objective function. One common method to minimize communication costs is by using local stochastic gradient descent (SGD), where the local device performs several update steps with their local data before averaging the model weights $\mathcal{W}$. FedAvg [49] is a popular algorithm that adopts this approach.

3.3. Generate Sub-network Architectures

To prune a model, we need to first generate the corresponding sub-network architectures. To achieve this goal, we use a binary vector $\mathbf{a} \in \{0, 1\}$ to represent whether to keep or prune a channel. To facilitate the learning of the sub-network architecture, we use a hypernetwork [16] (HN) to generate the architecture vector $\mathbf{a}$:

$$\mathbf{a} = \text{HN}(e; \theta_{\text{HN}}), \tag{2}$$

where e is the network embedding of the corresponding device or server, which will be discussed later, and θ_{HN} is the parameters of the HN. We use Straight-Through Gumbel-Sigmoid [25] to enable gradient calculation for the HN. To control the pruning of each channel, we apply $\mathbf{a}$ to the feature map of each layer:

$$\hat{\mathcal{F}}_l = \mathbf{a}_l \odot \mathcal{F}_l, \tag{3}$$

where $\hat{\mathcal{F}}_l$ is the feature map after applying $\mathbf{a}_l$ (the architecture vector of lth layer). Note that we insert $\mathbf{a}_l$ after normalization and activation layers, which correspond to control the output channels of the previous convolution layer and input channels of the next convolution layer.

An alternative approach to control pruning is to add learnable parameters for each channel. However, this approach presents challenges in the context of FL. If we only train a single sub-network for compression, local parameters must be accumulated on the server. Using individual learnable parameters for each channel may result in significantly different parameters across devices due to the non-iid setting, rendering the final parameters meaningless (often close to 0.5 before binarization). Additionally, if we aim to learn device-wise sub-networks for pruning, we would need to train N sets of parameters for all devices, making it unclear how to share knowledge between devices.

3.4. Architecture Embedding for Server and Device Side Sub-networks

Our method aims to find the appropriate server-side sub-network and device-side sub-networks. The server-side sub-network serves as the weight bank for device-side sub-networks. In addition, it reduces the communication and training costs at the finetuning stage and alleviates the memory burden on each device. Device-side sub-networks are used to meet the resource constraint of each device at the inference time, assuming the training and test data distribution on each device are similar. Using HN potentially provides a unique opportunity to share knowledge between server-side and device-side sub-networks. To achieve this, we introduce an embedding layer to produce the embedding for each sub-network:

$$e_n = \text{Emb}(n; \theta_{\text{Emb}}),\ n = 0, \cdots, N, \tag{4}$$

where θ_{Emb} is the parameters of the embedding layer Emb, and n is the index for each device. In addition, we let $n = 0$ represent the embedding for the server-side sub-network. By putting Eq. 2 and Eq. 4 together, we can generate the server-side sub-network and device-side sub-networks by using:

$$\begin{aligned} \mathbf{a}^0 &= \text{HN}(\text{Emb}(0; \theta_{\text{Emb}}); \theta_{\text{HN}}), \\ \mathbf{a}^n &= \text{HN}(\text{Emb}(n; \theta_{\text{Emb}}); \theta_{\text{HN}}),\ n = 1, \cdots, N, \end{aligned} \tag{5}$$

Algorithm 1: Learning Server-side and Device-side Sub-networks

```
Input: $D_n$, $D_n^{\mathbf{a}}$ ,$p_s$, $p_d^n$, $\lambda$, $S$, $K$, $r_{\mathcal{W}}$, $r_\theta$, $r_{\text{HN}}$
Initialization: $k_{\text{HN}} = 0$.
broadcast the current state of the CNN
for $k := 1$ to $K$ do
    /* Training the CNN. Freeze $\theta$ of the HN. */
    for For each device in parallel do
        1. Calculate gradients w.r.t to the loss function defined in Eq. 7.
        2. Update local CNN weights using the preferred optimizer.
    end
    /* Server updates of the CNN. */
    3. if $k$ % $r_{\mathcal{W}}$ = 0 then
        Randomly sample $S$ devices,
        average states of the CNN and broadcasts the updated states.
    /* Training the HN. Freeze $\mathcal{W}$ of the CNN. */
    if $k$ % $r_{HN}$ = 0 then
        for For each device in parallel do
            1. Calculate gradients w.r.t $\theta$ given the loss function defined in Eq. 6.
            2. Update local HN weights (including Emb) using the preferred optimizer.
        end
        /* Server updates of the HN. */
        3. if $k_{HN}$ % $r_\theta$ = 0 then
            Randomly sample $S$ devices,
            average states of the HN and broadcasts the updated states.
        4. $k_{\text{HN}}$ = $k_{\text{HN}}$ + 1
end
Pruning the model with resulting $\mathbf{a}^0$, and fine-tuning it.
```

where $\mathbf{a}^0$ is the server-side sub-network and $\mathbf{a}^n$ are device-side sub-networks. The embedding layer is used more frequently in natural language processing [50], but it well suits our task since it can covert the device id into a corresponding sub-network by combining Emb and HN.

3.5. Channel Pruning for Federated Learning

Given the aforementioned settings, we can now formally introduce the objective function for channel pruning. The channel pruning problem can be viewed as a constrained optimization problem, where the constraint is used to control the computational resource of the sub-network. The channel pruning objective function can be formulated as follows:

$$\min_\theta \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}(\mathcal{W}, D_n^{\mathbf{a}}; \mathbf{a}^0 \odot \mathbf{a}^n) \tag{6}$$

$$+ \lambda[(\mathcal{R}(T(\mathbf{a}^0), p_s T_{\text{total}}) + \frac{1}{N}\sum_{n=1}^{N} \mathcal{R}(T(\mathbf{a}^0 \odot \mathbf{a}^n), p_d^n T_{\text{total}})],$$

where θ contains both θ_{HN} and θ_{Emb}, $\mathbf{a}^0$ and $\mathbf{a}^n$ are generated by using Eq. 5, $D_n^{\mathbf{a}}$ is a subset of the local datasets D_n, $\mathcal{R}$ is the regularization loss to control the FLOPs of the sub-network, $p_s \in (0,1]$ is a predefined hyperparameter to control the preserved FLOPs of the server-side sub-network, $p_d^n \in (0,1],\ n = 1, \cdots, N$ are also predefined hyperparameters to control the FLOPs of sub-networks on each device, $T(\mathbf{a}^0)$ or $T(\mathbf{a}^0 \odot \mathbf{a}^n)$ is the current FLOPs decided by the sub-network architecture $\mathbf{a}^0$ or $\mathbf{a}^0 \odot \mathbf{a}^n$, and T_{total} is the total FLOPs of the CNN. The FLOPs constraint $\mathcal{R}(x,y)$ is generally a regression problem, but regular regression loss functions, like MAE and MSE, can hardly push $\mathcal{R}$ to near zero values. We let $\mathcal{R}(x,y) = \log(\frac{\max(x,y)}{y})$ to push $\mathcal{R}$ to be close to 0. In addition, we explicitly require that if a channel is pruned by the server-side sub-network, the corresponding device-side sub-networks should not update the corresponding position, and the detail is shown in the supplementary materials.

We perform iterative updates between model weights and the sub-network architectures. When updating model weights, we use the following equation:

$$\min_{\mathcal{W}} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathcal{W}, D_n; \mathbf{a}^0 \odot \mathbf{a}^n). \tag{7}$$

When training model weights, we freeze the sub-networks generated by the HN, and when training the HN, we also freeze $\mathcal{W}$.

The overview of our method is shown in Fig. 1. The algorithm of training our method for **one epoch** is shown in Alg. 1. In Alg. 1 D_n and $D_n^{\mathbf{a}}$ are local datasets and their sub-set for training the HN. λ is the hyperparameter to control FLOPs constraints, S is the number of sampled devices, K is the number of iterations within one epoch, $r_{\mathcal{W}}$ is the state average interval for the CNN, r_θ is the state average interval for the HN, r_{HN} decides the frequency of training the HN. To control the communication and the additional training costs, we introduce three hyperparameters: $r_{\mathcal{W}}$, r_θ and r_{HN}. $r_{\mathcal{W}}$, r_θ controls the communication costs for training the model and the HN. Larger $r_{\mathcal{W}}$ and r_θ will reduce the communication costs, but it may also negatively affect the quality of the final model and generated sub-networks under the FL setting. r_{HN} controls the overall training costs brought by HN. Similarly, larger r_{HN} results in smaller additional training costs, but it makes the training of HN harder since the difference of model weights is larger between consecutive training iterations of the HN. We follow Mime [31] for averaging states of the optimizers.

3.6. Theoretical Guarantee of Convergence

The objective of channel pruning is finding an optimal sub-network such that the FLOPs constraint is satisfied and the model performance is maximized. In fact, channel pruning in our setting can be viewed as a federated bilevel optimization problem [63, 69]. More formally, we combine Eq.(6) and Eq. (7) to have:

$$\begin{aligned}
&\min_{\theta} h(\theta) := \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathcal{W}_\theta, D_n^{\mathbf{a}}; \mathbf{a}^0 \odot \mathbf{a}^n) \\
&+ \lambda[(\mathcal{R}(T(\mathbf{a}^0), p_s T_{\text{total}}) + \frac{1}{N} \sum_{n=1}^{N} \mathcal{R}(T(\mathbf{a}^0 \odot \mathbf{a}^n), p_d^n T_{\text{total}})], \\
&s.t. \mathcal{W}_\theta = \arg\min_{\mathcal{W}} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathcal{W}, D_n; \mathbf{a}^0 \odot \mathbf{a}^n).
\end{aligned} \tag{8}$$

From a bilevel's perspective, we do channel pruning by iteratively performing the following steps until convergence: for a given sub-network structure from the HN and Emb, we first find the optimal model weight $\mathcal{W}$(solving the lower level problem in Eq. (8)); then we optimize the sub-network based on this optimal model weight(solving the upper level problem in Eq. (8)). Finding the optimal model weight is expensive, especially for modern deep neural networks; we instead optimize the HN and Emb weights θ and the model weight $\mathcal{W}$ alternatively as in Alg. 1. Furthermore, the gradient *w.r.t* the sub-network structure includes both a direct part, which is the direct gradient *w.r.t* θ, and an indirect part due to $\mathcal{W}_\theta$ is a function of θ (the minimizer of the lower level problem). However, the indirect gradient is expensive to evaluate and leads to minor empirical improvement in practice, so we only consider the direct gradient when updating θ in Alg. 1. The convergence of our alternative update method is guaranteed under mild assumptions [26, 73] as stated in Theorem 3.1 below:

Theorem 3.1. *Suppose we choose the upper level learning rate η and the lower level learning rate γ as:*

$$\eta = \min\left\{\frac{1}{4Lr_\theta}, \left(\frac{2bN\Delta_\theta}{K_{HN}L\sigma^2}\right)^{1/2}\right\},$$

and

$$\gamma = \min\left\{\frac{1}{4L}, \left(\frac{\lambda bN}{K_{HN}L^2\sigma^2}\right)^{1/2}\right\},$$

then we have:

$$\frac{1}{K_{HN}} \sum_{k_{HN}=0}^{K_{HN}-1} \mathbb{E}\|\nabla h(\bar{\theta}_{k_{HN}})\|^2 = O\left(\frac{1}{(bNK_{HN})^{1/2}}\right)$$

where b is the mini-batch size, N is the number of devices, and K_{HN} is the number of update steps to the upper level variable θ.

As stated in Theorem 3.1, our algorithm converges to a stationary point of Eq. (8), with a convergence rate of $O(K_{HN}^{-0.5})$. Furthermore, the algorithm achieves linear speed up *w.r.t* the number of devices and mini-batch size.

Method	Dataset	Architecture	Base Acc	Δ-Acc	Acc	↓ FLOPs (D)	↓ FLOPs (S)
Filter Pruning [38]	CIFAR-10	ResNet-56	91.22%	-0.93%	90.29%	50%	50%
FedOSP				-0.28%	90.94%	50%	50%
FedILP				-0.08%	91.14%	50%	50%
DWNP				**+0.66%**	**91.88%**	50%	20%
Filter Pruning [38]	CIFAR-100	ResNet-18	66.57%	-1.31%	65.26%	50%	50%
FedOSP				-0.61%	65.96%	50%	50%
FedILP				-0.20%	66.37%	50%	50%
DWNP				**+1.74%**	**68.31%**	50%	20%
Filter Pruning [38]				-2.52%	64.05%	70%	70%
FedOSP				-1.79%	64.78%	70%	70%
FedILP				-1.44%	65.13%	70%	70%
DWNP				**+0.05%**	**66.62%**	70%	50%
Filter Pruning [38]		ResNet-34	69.05%	-1.22%	67.83%	50%	50%
FedOSP				-0.29%	68.76%	50%	50%
FedILP				+0.44%	69.49%	50%	50%
DWNP				**+2.17%**	**71.72%**	50%	20%
FedILP		MobileNet-V2	66.76%	-0.22%	66.64%	48%	48%
DWNP				**+1.46%**	**68.22%**	48%	20%

Table 1. Results of CIFAR-10 and CIFAR-100. 'Base Acc' represents the baseline training accuracy. 'Δ-Acc' represents the accuracy changes before and after pruning. 'Acc' represents the accuracy after pruning. '↓ FLOPs (D)' and '↓ FLOPs (S)' represent the pruned FLOPs of device-side and server-side sub-networks.

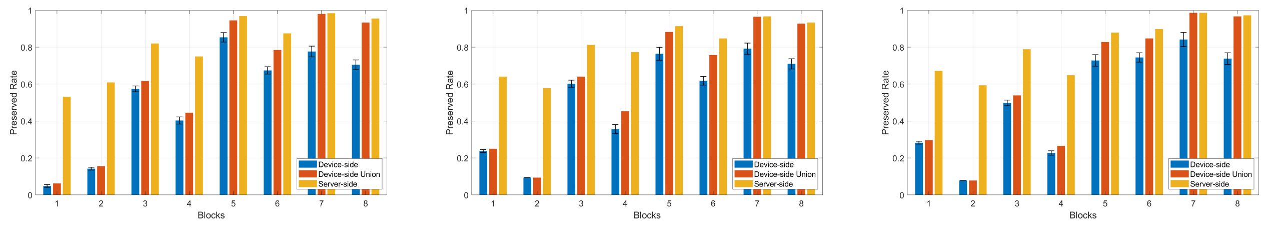
Architecture	Method	Base Top-1 Acc	Base Top-5 Acc	Δ Top-1 Acc	Δ Top-5 Acc	↓ FLOPs (D)	↓ FLOPs (S)
ResNet-18	Filter Pruning [38]	54.99%	78.60%	-1.01%	-0.33%	50%	50%
	FedOSP			-0.18%	+0.48%	50%	50%
	FedILP			+0.07%	+0.65%	50%	50%
	DWNP			**+1.06%**	**+1.10%**	50%	20%
ResNet-34	Filter Pruning [38]	56.32%	79.37%	-0.91%	-0.21%	50%	50%
	FedOSP			-0.20%	+0.21%	50%	50%
	FedILP			-0.03%	+0.34%	50%	50%
	DWNP			**+0.80%**	**+0.74%**	50%	20%

Table 2. Comparison results on TinyImageNet with ResNet-18/34. 'Base Top-1/5' represents the baseline training Top-1/5 accuracy. 'Δ Top-1/5 Acc' represents the Top-1/5 accuracy changes before and after pruning.

4. Experiments

4.1. Settings

Datasets and Models. We use CIFAR-10 [34], CIFAR-100 [34], and TinyImageNet [6, 36] to evaluate the performance of our method. Our method uses p_s and p_d^n to control the FLOPs for the server and each device. In the experiment section, we assume p_d^n has the same value for different devices for a fair comparison with other methods. The detailed choices of p_s and p_d^n are listed in supplementary materials. We choose ResNets [19] and MobileNet-V2 [59] for comparison. For CIFAR-10, we compare our method with other baselines on ResNet-56. For CIFAR-100, we compare our method with other baselines on ResNet-18, ResNet-34, and MobileNet-V2. For TinyImageNet, ResNet-18 and ResNet-34 are used for comparisons. To reduce the negative effects caused by batch normalization layers, we replace batch normalization with layer normalization [2], which has been used frequently in recent designs of vision transformers [7] and CNNs [47]. For the main experiments, we consider N = 10 devices. We use the Dirichlet distribution with $\alpha = 0.5$, as described in [48], to create non-iid partitions on the devices for all datasets. Other settings of N and α are also verified for specific models and datasets. As described in section 3, we assume the training and test datasets on each device are similar. To accomplish this, we apply a random permutation to the samples drawn from the Dirichlet distribution for the training dataset and then split the test dataset based on the permuted samples. As a result, the training and test datasets distributions on each device are similar but not the same. More details are given in the supplementary materials. **Baselines.** In addition to the proposed method, we also build three baselines from the literature on channel pruning. **(1) Filter Pruning**: we directly adapt the Filter Pruning [38] to the FL setting, where there are no communication costs for pruning. In this setting, pruning is purely based on the channel norm of the weights. **(2) FedOSP** (**Fed**erated **O**ne-**S**hot **P**runing): this baseline can be seen as an improved version of channel pruning methods with differentiable gates [11, 32, 76] in the one-shot pruning setting. In this setting, we use the HN to generate one sub-network for all devices. The HN is learned in a one-shot setting when model weights are frozen. **(3) FedILP** (**Fed**erated **I**terative-**L**earning and **P**runing): this baseline can be seen as the simplified version of our method without device-side sub-networks. Through the experiment section, our method is abbreviated as **DWNP** (**D**evice-**W**ise **N**etwork **P**runing). For all settings, we report the mean



(a) $\alpha = 0.05$ (b) $\alpha = 0.10$ (c) $\alpha = 0.50$

Figure 2. The layer-wise pruning rates for device-side sub-networks, the union of device-side sub-networks, and the server-side sub-network with different α. The union of device-side sub-networks represents the union of kept channels from all devices.

Method	$N = 10$		$N = 25$		$N = 50$	
	Base	Acc	Base	Acc	Base	Acc
FedILP	66.57%	66.37% -0.20%	65.86%	65.37% -0.49%	65.13%	64.88% -0.25%
DWNP		68.31% **+1.74%**		68.09% **+2.23%**		67.58% **+2.45%**

Table 3. Performance of pruned models given different numbers of devices N with ResNet-18 on CIFAR-100.

Method	Architecture	$\alpha = 0.5$		$\alpha = 0.1$		$\alpha = 0.05$	
		Base	Acc	Base	Acc	Base	Acc
FedILP	ResNet-18	66.57%	66.37% -0.20%	63.22%	62.77% -0.55%	61.61%	60.50% -1.11%
DWNP			68.31% **+1.74%**		64.51% **+1.29%**		62.59% **+0.98%**
FedILP	ResNet-34	69.05%	69.49% +0.44%	66.77%	66.46% -0.31%	64.52%	63.54% -0.98%
DWNP			71.72% **+2.17%**		68.20% **+1.43%**		65.53% **+1.01%**

Table 4. Performance of pruned models given different choices of α with ResNet-18/34 on CIFAR-100.

results across three runs.

Training Settings. We describe the hyperparameters in Alg. 1 in this section. For all methods, we let $S = N$, $\lambda = 2.0$, $r_{\mathcal{W}} = 5$, $r_{\theta} = 2$ and $r_{\text{HN}} = 10$. K is the number of iterations for training one epoch. For Filter Pruning and FedOSP, we train a base model for 200 epochs, and this base model also servers as the baseline model in Tab. 1 and Tab. 2. For FedILP and DWNP, we train the model and the hypernetwork from scratch for 200 epochs. For FedIPL and DWNP, we start the training of the hypernetwork after $\frac{1}{4}$ of the total training epochs, which avoids misleading pruning results when weights are not properly trained. We finetune the model for 200 epochs for all methods to recover the performance. For each local dataset, we sample 10% of the training samples to construct $D_n^{\mathbf{a}}$. When updating the local $\mathcal{W}$, we use SGD with momentum 0.9 and a start learning rate 0.1. When updating the local θ, we use Adam [33] with a start learning rate of 10^{-3}. Other training details are shown in the supplementary materials.

4.2. Results

CIFAR-10/CIFAR-100. We tested different settings on CIFAR-10 and CIFAR-100 and found that our method DWNP consistently achieved the best performance across different model architectures and pruning rates. Specifically, DWNP outperformed the original model by 0.66%, 1.74%, 2.17%, and 1.46% for ResNet-56, ResNet-18, ResNet-34, and MobileNet-V2, respectively. This demonstrated that the design of device-wise sub-networks is beneficial for achieving a good trade-off for channel pruning under the federated learning setup. Our method even surpassed the original model when pruning 70% of FLOPs on ResNet-18. The relative ranking of other baselines is Filter Pruning, FedOSP, and FedILP. Our method achieved a prominent trade-off between performance and computational costs on more complex datasets, like CIFAR-100, with an improvement of 0.05%~2.17% over the original model when pruning 50% FLOPs or more. The performance of our method on MobileNet-V2 demonstrated that it could be seamlessly extended to lightweight models.

TinyImageNet. We present the results of ResNet-18 and ResNet-34 on TinyImageNet in Tab. 2. DWNP is 1.06%/1.10% better than the original model regarding the Top-1/5 accuracy for ResNet-18. For ResNet-34, the advantage is 0.80%/0.74% regarding the Top-1/5 accuracy. The advantage of our method compared to other baselines is still obvious, which ranges from 1.13% $\sim$ 2.07% and 0.45% $\sim$ 1.75% for Δ Top-1/5 accuracy for ResNet-18. We have similar observations for ResNet-34.

Across all settings, FedILP often performs better than FedOSP, indicating that learning model weights and architectures simultaneously are beneficial, as explained in section 3.6. In general, data-driven approaches perform better than pruning methods based on channel norms, suggesting that local data distributions should be considered explicitly when pruning under the FL setting. Indeed, the performance gain of DWNP is not free. For the server-side sub-network, the FLOPs reduction for DWNP is much smaller than other methods, and DWNP has to occupy more storage space on the server.

Other Settings. To verify whether our method can perform well in other settings, we change N and α to create different FL settings. In the first experiment, we use ResNet-18 on CIFAR-100 to verify whether our method can achieve sim-

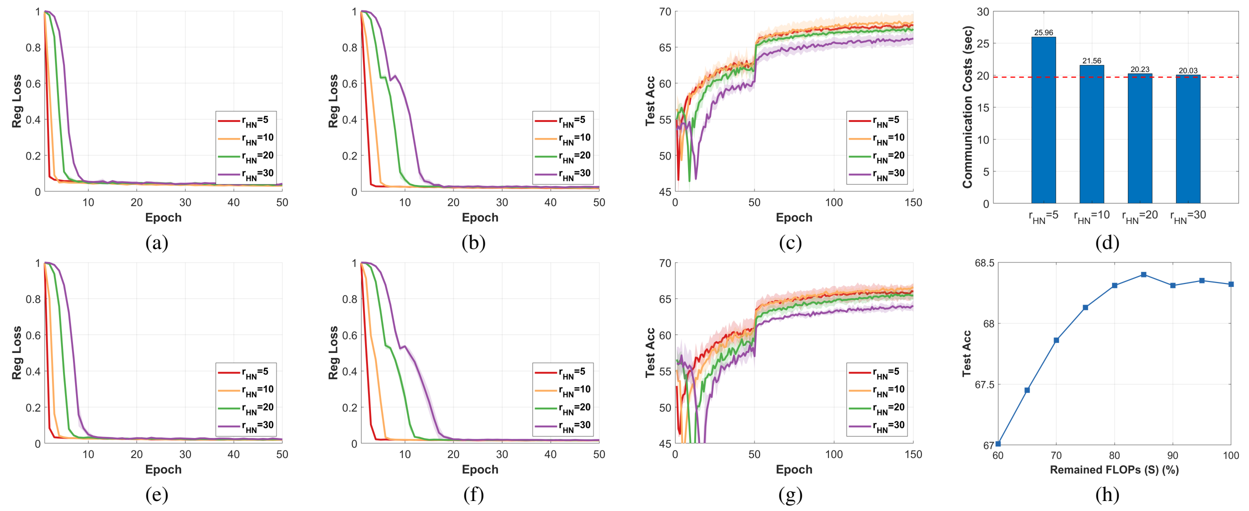


Figure 3. (a, e): normalized FLOPs regularization loss values for the server-side sub-network. (b, f): normalized FLOPs regularization loss values for device-side sub-networks. (d, h): test accuracy given different choices of r_{HN}. (d): communication costs. (h): trade-off between server-side and device-side sub-networks. Experiments are conducted on CIFAR-100 with ResNet-18 and when pruning 50% FLOPs (a,b,c) and 70% FLOPs (f,g,h) on devices.

ilar performance when changing the number of devices N. From Tab. 3, we can see the Δ-Acc is increased given more devices, which shows that our method is more resilient when increasing the number of devices N. On the other hand, the Δ-Acc of FedILP is similar or worse when increasing the number of devices, probably because the learning of the sub-network becomes harder when increasing N. In Tab. 4, we show the results when changing α on ResNet-18/34. A smaller α represents more diverse local data distributions and is often harder for model training. The table shows that both DWNP and FedILP are affected by decreasing α. However, DWNP can still maintain a positive performance gain for both ResNet-18/34. We plot the layer-wise pruning rate for channels with different α in Fig. 2. It can be seen that the sub-network architecture changes when changing local data distributions. For high heterogeneity ($\alpha = 0.05$), DWNP prefers to perverse more later layer channels, which is plausible because feature maps of later layers are more diverse on each device. In addition, the early stages of the model are not well utilized by device-side sub-networks. On the one hand, it is reasonable since CNNs tend to learn uniform representations from early stages. On the other hand, maybe we can add constraints to encourage the utilization of early stages or adjust the server-side sub-network so that it can be better used.

Detailed Analysis. We examine how r_{HN} changes the training dynamics during the optimization process. The training of model weights is not the focus of our paper, so we did not study $r_{\mathcal{W}}$. The effect of r_θ is not obvious compared to r_{HN}. We present our study in Fig. 3. We plot the first 50 epochs for regularization loss values after the training of HN begins. As described in the settings 4.1, the training of HN starts after 50 epochs of model weights training. We test 4 settings of r_{HN}: $\{5,10,20,30\}$. In short, our method performs well when $r_{\text{HN}} \leq 10$. We can see an obvious performance drop when $r_{\text{HN}} = 30$. We also plot the overall communication costs for $\mathcal{W}$ and θ in Fig. 3d, and the red dashed line represents the costs for $\mathcal{W}$ only. For $r_{\text{HN}} \geq 10$, the communication overhead from training the HN becomes marginal. As a result, $r_{\text{HN}} = 10$ provides a good trade-off between performance and additional communication costs. In Fig. 3h, we show the trade-off between the model performance and the server-side FLOPs when pruning 50% of FLOPs on devices with ResNet-18 on CIFAR-100. We can see that our method can maintain a good performance when the remained server-side FLOPs are larger than 75%.

5. Conclusion

In this paper, we proposed a new channel pruning method under the Federated Learning settings. Specifically, we generate device-side sub-networks from the server-side sub-network through a hypernetwork and a network embedding layer for device-wise pruning. Our method can be optimized in an end-to-end differentiable fashion, which is very efficient. In addition, the extra communication costs and training costs for the hypernetwork and the embedding layer can be easily controlled using only two hyperparameters. Furthermore, we establish a theoretical guarantee of convergence, affirming that our method converges to a stationary point. Our method achieves competitive performance on CIFAR-10, CIFAR-100, and TinyImageNet datasets with ResNets and MobileNet-V2.

References

[1] Jan Philipp Albrecht. How the gdpr will change the world. *Eur. Data Prot. L. Rev.*, 2:287, 2016. 1

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 6

[3] Sameer Bibikar, Haris Vikalo, Zhangyang Wang, and Xiaohan Chen. Federated dynamic sparse training: Computing less, communicating less, yet learning better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6080–6088, 2022. 3

[4] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014. 6

[5] Yubei Chen Chun-Hsiao Yeh. IN100pytorch: Pytorch implementation: Training resnets on imagenet-100. `https://github.com/danielchyeh/ImageNet-100-Pytorch`, 2022. 7

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009. 6

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*. 6

[8] Alireza Ganjdanesh*, Shangqian Gao*, and Heng Huang. Interpretations steered network pruning via amortized inferred saliency maps. In *European Conference on Computer Vision*, pages 278–296. Springer, 2022. 2

[9] Alireza Ganjdanesh, Shangqian Gao, and Heng Huang. Eff-conv: efficient learning of kernel sizes for convolution layers of cnns. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7604–7612, 2023.

[10] Alireza Ganjdanesh*, Shangqian Gao*, Hirad Alipanah, and Heng Huang. Compressing image-to-image translation gans using local density structures on their learned manifold. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

[11] Shangqian Gao, Feihu Huang, Jian Pei, and Heng Huang. Discrete model compression with resource constraint for deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1899–1908, 2020. 2, 6

[12] Shangqian Gao, Feihu Huang, Yanfu Zhang, and Heng Huang. Disentangled differentiable network pruning. In *European Conference on Computer Vision*, pages 328–345. Springer, 2022.

[13] Shangqian Gao, Burak Uzkent, Yilin Shen, Heng Huang, and Hongxia Jin. Learning to jointly share and prune weights for grounding based vision and language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[14] Shangqian Gao, Zeyu Zhang, Yanfu Zhang, Feihu Huang, and Heng Huang. Structural alignment for network pruning through partial regularization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17402–17412, 2023. 2

[15] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients–how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*, 2020. 2

[16] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 2, 3

[17] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1

[18] 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. 1, 2

[19] 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. 2, 6

[20] Feihu Huang, Junyi Li, and Heng Huang. Compositional federated learning: Applications in distributionally robust averaging and meta learning. *arXiv preprint arXiv:2106.11264*, 2021. 2

[21] Hong Huang, Lan Zhang, Chaoyue Sun, Ruogu Fang, Xiaoyong Yuan, and Dapeng Wu. Fedtiny: Pruned federated learning towards specialized tiny models. *arXiv preprint arXiv:2212.01977*, 2022. 1, 3

[22] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018. 1, 2

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, pages 448–456. JMLR.org, 2015. 2

[24] Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Vladimir Braverman, Ion Stoica, and Raman Arora. Communication-efficient distributed sgd with sketching. *arXiv preprint arXiv:1903.04488*, 2019. 2

[25] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 4

[26] Kaiyi Ji, Junjie Yang, and Yingbin Liang. Provably faster algorithms for bilevel optimization and applications to meta-learning. *arXiv preprint arXiv:2010.07962*, 2020. 5

[27] Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K Leung, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 1, 3

[28] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019. 2

[29] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for on-device federated learning. *arXiv preprint arXiv:1910.06378*, 2019. 2

[30] 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*, pages 3252–3261. PMLR, 2019. 2

[31] Sai Praneeth Karimireddy, Martin Jaggi, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. Mime: Mimicking centralized stochastic algorithms in federated learning. *arXiv preprint arXiv:2008.03606*, 2020. 5

[32] Jaedeok Kim, Chiyoun Park, Hyun-Joo Jung, and Yoonsuck Choe. Plug-in, trainable gate for streamlining arbitrary neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. 2, 6

[33] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7

[34] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 6

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. 1

[36] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015. 6

[37] Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. Lotteryfl: empower edge intelligence with personalized and communication-efficient federated learning. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 68–79. IEEE, 2021. 1, 3

[38] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 1, 2, 6

[39] Junyi Li and Heng Huang. Faster secure data mining via distributed homomorphic encryption. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2706–2714, 2020. 2

[40] Junyi Li, Jian Pei, and Heng Huang. Communication-efficient robust federated learning with noisy labels. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 914–924, 2022. 3

[41] Junyi Li, Feihu Huang, and Heng Huang. Communication-efficient federated bilevel optimization with global and local lower level problems. *Advances in Neural Information Processing Systems*, 36, 2023. 3

[42] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 2021. 1, 2

[43] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning*, pages 6357–6368. PMLR, 2021. 2

[44] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, 2019. 2

[45] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017. 2

[46] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017. 2

[47] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022. 6

[48] Mi Luo, Fei Chen, Dapeng Hu, Yifan Zhang, Jian Liang, and Jiashi Feng. No fear of heterogeneity: Classifier calibration for federated learning with non-iid data. *Advances in Neural Information Processing Systems*, 34:5972–5984, 2021. 6

[49] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017. 1, 2, 3

[50] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. 4

[51] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. In *International Conference on Machine Learning*, pages 4615–4625. PMLR, 2019. 2

[52] Muhammad Tahir Munir, Muhammad Mustansar Saeed, Mahad Ali, Zafar Ayyub Qazi, and Ihsan Ayyub Qazi. Fedprune: Towards inclusive federated learning. *arXiv preprint arXiv:2110.14205*, 2021. 1, 3

[53] Karthik Nandakumar, Nalini Ratha, Sharath Pankanti, and Shai Halevi. Towards deep neural network training on encrypted data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019. 2

[54] Xinchi Qiu, Javier Fernandez-Marques, Pedro PB Gusmao, Yan Gao, Titouan Parcollet, and Nicholas Donald Lane. Zerofl: Efficient on-device training for federated learning with local sparsity. *arXiv preprint arXiv:2208.02507*, 2022. 3

[55] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016. 1

[56] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 1

[57] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. Fetchsgd: Communication-efficient federated learning with sketching. In *International Conference on Machine Learning*, pages 8253–8265. PMLR, 2020. 2

[58] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 3, 2018. 2

[59] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 2, 6

[60] Rulin Shao, Hui Liu, and Dianbo Liu. Privacy preserving stochastic channel-based federated learning with neural network pruning. *arXiv preprint arXiv:1910.02115*, 2019. 1, 3

[61] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014. 1

[62] Hannah Smith, Zeyu Zhang, John Culnan, and Peter Jansen. ScienceExamCER: A high-density fine-grained science-domain corpus for common entity recognition. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 4529–4546, Marseille, France, 2020. European Language Resources Association. 2

[63] Mikhail Solodov. An explicit descent method for bilevel convex optimization. *Journal of Convex Analysis*, 14(2):227, 2007. 5

[64] Sebastian U Stich. Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018. 2

[65] Davoud Ataee Tarzanagh, Mingchen Li, Christos Thrampoulidis, and Samet Oymak. Fednest: Federated bilevel, minimax, and compositional optimization. In *International Conference on Machine Learning*, pages 21146–21179. PMLR, 2022. 3

[66] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proc. Priv. Enhancing Technol.*, 2019(3):26–49, 2019. 2

[67] Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6151–6162, Online, 2020. Association for Computational Linguistics. 2

[68] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. *arXiv preprint arXiv:1705.07878*, 2017. 2

[69] Ralph A Willoughby. Solutions of ill-posed problems (an tikhonov and vy arsenin). *SIAM Review*, 21(2):266, 1979. 5

[70] Blake Woodworth, Kumar Kshitij Patel, Sebastian Stich, Zhen Dai, Brian Bullins, Brendan Mcmahan, Ohad Shamir, and Nathan Srebro. Is local sgd better than minibatch sgd? In *International Conference on Machine Learning*, pages 10334–10343. PMLR, 2020. 1

[71] Peiyao Xiao and Kaiyi Ji. Communication-efficient federated hypergradient computation via aggregated iterative differentiation. In *International Conference on Machine Learning*, pages 38059–38086. PMLR, 2023. 3

[72] Dongfang Xu, Zeyu Zhang, and Steven Bethard. A generate-and-rank framework with semantic type regularization for biomedical concept normalization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8452–8464, Online, 2020. Association for Computational Linguistics. 2

[73] Junjie Yang, Kaiyi Ji, and Yingbin Liang. Provably faster algorithms for bilevel optimization. *arXiv preprint arXiv:2106.04692*, 2021. 5

[74] 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):1–19, 2019. 1

[75] Yifan Yang, Peiyao Xiao, and Kaiyi Ji. Simfbo: Towards simple, flexible and communication-efficient federated bilevel learning. *Advances in Neural Information Processing Systems*, 36, 2023. 3

[76] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 2130–2141, 2019. 1, 2, 6

[77] Zeyu Zhang and Steven Bethard. Improving toponym resolution with better candidate generation, transformer-based reranking, and two-stage resolution. In *Proceedings of the 12th Joint Conference on Lexical and Computational Semantics (*SEM 2023)*, pages 48–60, Toronto, Canada, 2023. Association for Computational Linguistics. 2

[78] Zeyu Zhang, Thuy Vu, and Alessandro Moschitti. Joint models for answer verification in question answering systems. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3252–3262, Online, 2021. Association for Computational Linguistics.

[79] Zeyu Zhang, Thuy Vu, Sunil Gandhi, Ankit Chadha, and Alessandro Moschitti. Wdrass: A web-scale dataset for document retrieval and answer sentence selection. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, page 4707–4711, New York, NY, USA, 2022. Association for Computing Machinery.

[80] Zeyu Zhang, Thuy Vu, and Alessandro Moschitti. In situ answer sentence selection at web-scale. *arXiv preprint arXiv:2201.05984*, 2022.

[81] Zeyu Zhang, Thuy Vu, and Alessandro Moschitti. Double retrieval and ranking for accurate question answering. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1751–1762, Dubrovnik, Croatia, 2023. Association for Computational Linguistics. 2

[82] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018. 2

A. Proof for the convergence of Algorithm 1

In this section, we study the convergence rate of Algorithm 1. We first simplify the notations of objective functions: $\mathcal{L}^{(n)}(\mathcal{W};\theta) = \mathcal{L}(\mathcal{W}, D_n; \mathbf{a}^0 \odot \mathbf{a}^n)$ and

$$h^{(n)}(\mathcal{W},\theta) = \mathcal{L}(\mathcal{W}_\theta, D_n^{\mathbf{a}}; \mathbf{a}^0 \odot \mathbf{a}^n) + \lambda[(\mathcal{R}(T(\mathbf{a}^0), p_s T_{\text{total}}) + \mathcal{R}(T(\mathbf{a}^0 \odot \mathbf{a}^n), p_d^n T_{\text{total}})]$$

Next, we make some mild assumptions:

Assumption A.1. The function $h^{(n)}(\mathcal{W},\theta)$ is possibly non-convex, L-Lipschitz; $\mathcal{L}^{(n)}(\mathcal{W},\theta)$ is L-Lipschitz, μ-strongly convex *w.r.t.* $\mathcal{W}$ for any given θ;

Assumption A.2. We have an unbiased stochastic first order derivative oracle with bounded variance σ^2;

Assumption A.3. For any $m, j \in [N]$ and $z = (x, y)$, we have: $\|\nabla h^{(m)}(\mathcal{W},\theta) - \nabla h^{(j)}(\mathcal{W},\theta)\| \le \zeta$ and $\|\nabla \mathcal{L}^{(m)}(\mathcal{W},\theta) - \nabla \mathcal{L}^{(j)}(\mathcal{W},\theta)\| \le \bar{\zeta}$, where ζ, $\bar{\zeta}$ are constants.

We further assume the total number of update steps to $\mathcal{W}$ is K; the total number of update steps to θ is K_{HN}, and $K = r_{HN}K_{HN}$; the mini-batch size of stochastic query for both $\mathcal{W}$ and θ is b; the stochastic query to θ is denoted as G_k; we omit the gradient of $\mathcal{W}_\theta$ *w.r.t.* θ; we use $\bar{\theta}$ and $\bar{\mathcal{W}}$ to denotes the average of θ and $\mathcal{W}$ across devices. We denote:

$$\tilde{\mathcal{W}}_{\bar{\theta}} = \arg\min_{\mathcal{W}} \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}^{(n)}(\mathcal{W};\theta^{(n)}),\ \mathcal{W}_{\bar{\theta}} = \arg\min_{\mathcal{W}} \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}^{(n)}(\mathcal{W};\bar{\theta}).$$

since we make infrequent updates to θ, we omit the drift of the lower level solution caused by θ update, in other words, we assume $\tilde{\mathcal{W}}_{\bar{\theta}} \approx \mathcal{W}_{\bar{\theta}}$.

A.1. Useful Lemmas

Lemma A.4. *For $k \in [r_{HN}k_{HN}, r_{HN}k_{HN} + r_{HN} - 1]$, we have:*

$$\mathbb{E}\left\|\bar{\mathcal{W}}_{r_{HN}k_{HN}+r_{HN}} - \mathcal{W}_{\bar{\theta}_{k_{HN}}}\right\|^2 \le (1-\lambda\gamma)^{r_{HN}}\mathbb{E}\left\|\bar{\mathcal{W}}_{r_{HN}k_{HN}} - \mathcal{W}_{\bar{\theta}_{k_{HN}}}\right\|^2 + \frac{1}{\lambda}\left(\frac{\gamma\sigma^2}{bN} + 12Lr_{\mathcal{W}}\sigma^2\gamma^2 + 12Lr_{\mathcal{W}}^2\gamma^2\bar{\zeta}^2\right)$$

where the expectation is w.r.t the stochasticity of the algorithm.

Proof. For ease of notation, we denote $\mathcal{W}^* = \mathcal{W}_{\bar{\theta}_{k_{HN}}}$ in the proof, and follow Lemma 15 and Lemma 16 in [70], for $k \in [r_{HN}k_{HN}, r_{HN}k_{HN} + r_{HN} - 1]$, we have:

$$\mathbb{E}\left\|\bar{\mathcal{W}}_{k+1} - \mathcal{W}^*\right\|^2 \le (1-\lambda\gamma)\mathbb{E}\left\|\bar{\mathcal{W}}_k - \mathcal{W}^*\right\|^2 + \frac{\gamma^2\sigma^2}{bN} + 12Lr_{\mathcal{W}}\sigma^2\gamma^3 + 12Lr_{\mathcal{W}}^2\bar{\zeta}^2\gamma^3$$

Telescope for $k \in [r_{HN}k_{HN}, r_{HN}k_{HN} + r_{HN} - 1]$, we have:

$$\begin{aligned}\mathbb{E}\left\|\bar{\mathcal{W}}_{r_{HN}k_{HN}+r_{HN}} - \mathcal{W}^*\right\|^2 &\le (1-\lambda\gamma)^{r_{HN}}\mathbb{E}\left\|\bar{\mathcal{W}}_{r_{HN}k_{HN}} - \mathcal{W}^*\right\|^2 \\ &\quad + \sum_{k=r_{HN}k_{HN}}^{r_{HN}k_{HN}+r_{HN}} (1-\lambda\gamma)^{k-r_{HN}k_{HN}}\left(\frac{\gamma^2\sigma^2}{N} + 12Lr_{\mathcal{W}}\sigma^2\gamma^3 + 12Lr_{\mathcal{W}}^2\gamma^3\bar{\zeta}^2\right) \\ &\le (1-\lambda\gamma)^{r_{HN}}\mathbb{E}\left\|\bar{\mathcal{W}}_{r_{HN}k_{HN}} - \mathcal{W}^*\right\|^2 + \frac{1}{\lambda\gamma}\left(\frac{\gamma^2\sigma^2}{bN} + 12Lr_{\mathcal{W}}\sigma^2\gamma^3 + 12Lr_{\mathcal{W}}^2\gamma^3\bar{\zeta}^2\right)\end{aligned}$$

This completes the proof. □

Lemma A.5. *Suppose iterates $\theta_k^{(n)}$, $n \in [N]$, $k \in [K_{HN}]$ are generated from Algorithm 1, we have:*

$$(1 - 12L^2r_\theta^2\eta^2)\sum_{k=\tilde{k}_r}^{\tilde{k}_r+r_\theta-1}\sum_{n=1}^{N}\mathbb{E}\|\bar{\theta}_k - \theta_k^{(n)}\|^2 \le Nr_\theta^3\eta^2\left(\frac{2\sigma^2}{b} + 8\zeta^2 + \frac{12L^2}{\lambda}\left(\frac{\gamma\sigma^2}{bN} + 12Lr_{\mathcal{W}}\sigma^2\gamma^2 + 12Lr_{\mathcal{W}}^2\gamma^2\bar{\zeta}^2\right)\right)$$

where $\tilde{k}_r \in [K_{HN}]$ and $\tilde{k}_r\%r_\theta = 0$, and the expectation is w.r.t the stochasticity of the algorithm.

Proof. Suppose we denote $\tilde{k}_r \in [K_{HN}]$ where $\tilde{k}_r \% r_\theta = 0$. By Algorithm 1, we have $\theta_{\tilde{k}_r}^{(n)} = \bar{\theta}_{\tilde{k}_r}$. For $k \neq \tilde{k}_r$, we have:

$$\theta_k^{(n)} = \theta_{\tilde{k}_r}^{(n)} - \sum_{\ell=\tilde{k}_r}^{k-1} \eta G_\ell^{(n)} \quad \text{and} \quad \bar{\theta}_k = \bar{\theta}_{\tilde{k}_r} - \sum_{\ell=\tilde{k}_r}^{k-1} \eta \bar{G}_\ell.$$

So we have:

$$\sum_{n=1}^{N} \|\theta_k^{(n)} - \bar{\theta}_k\|^2 = \sum_{n=1}^{N} \Big\| \sum_{\ell=\tilde{k}_r}^{k-1} \left(\eta G_\ell^{(n)} - \eta \bar{G}_\ell\right) \Big\|^2 \leq r_\theta \sum_{\ell=\tilde{k}_r}^{k-1} \eta^2 \sum_{n=1}^{N} \|G_\ell^{(n)} - \bar{G}_\ell\|^2$$

where the equality uses the fact $\theta_{\tilde{k}_r}^{(n)} = \bar{\theta}_{\tilde{k}_r}$ for $k \in [K]$, the inequality uses the generalized triangle inequality. Next,

$$\begin{aligned}
&\mathbb{E}\Big\| G_\ell^{(n)} - \bar{G}_\ell \Big\|^2 \\
&= \mathbb{E}\Big\| \left(G_\ell^{(n)} - \mathbb{E}_\xi[G_\ell^{(n)}]\right) - \frac{1}{N}\sum_{j=1}^{N}\left(G_\ell^{(j)} - \mathbb{E}_\xi[G_\ell^{(j)}]\right) + \mathbb{E}_\xi[G_\ell^{(n)}] - \frac{1}{N}\sum_{j=1}^{N}\mathbb{E}_\xi[G_\ell^{(j)}] \Big\|^2 \\
&\leq 2\mathbb{E}\Big\| \left(G_\ell^{(n)} - \mathbb{E}_\xi[G_\ell^{(n)}]\right) - \frac{1}{N}\sum_{j=1}^{N}\left(G_\ell^{(j)} - \mathbb{E}_\xi[G_\ell^{(j)}]\right) \Big\|^2 + 2\mathbb{E}\Big\| \mathbb{E}_\xi[G_\ell^{(n)}] - \frac{1}{N}\sum_{j=1}^{N}\mathbb{E}_\xi[G_\ell^{(j)}] \Big\|^2 \\
&\overset{(a)}{\leq} 2\mathbb{E}\Big\| \left(G_\ell^{(n)} - \mathbb{E}_\xi[G_\ell^{(n)}]\right) \Big\|^2 + 2\mathbb{E}\Big\| \mathbb{E}_\xi[G_\ell^{(n)}] - \frac{1}{N}\sum_{j=1}^{N}\mathbb{E}_\xi[G_\ell^{(j)}] \Big\|^2 \\
&\leq 2\mathbb{E}\big\| \left(G_\ell^{(n)} - \mathbb{E}_\xi[G_\ell^{(n)}]\right) \big\|^2 + 4\mathbb{E}\Big\| \mathbb{E}_\xi[G_\ell^{(n)}] - \nabla h^{(n)}(\bar{\theta}_\ell) \Big\|^2 \\
&\qquad + 8\mathbb{E}\Big\| \nabla h^{(n)}(\bar{\theta}_\ell) - \nabla h(\bar{\theta}_\ell) \Big\|^2 + 8\mathbb{E}\Big\| \nabla h(\bar{\theta}_\ell) - \frac{1}{N}\sum_{j=1}^{N}\mathbb{E}_\xi[G_\ell^{(j)}] \Big\|^2 \\
&\overset{(b)}{\leq} \frac{2\sigma^2}{b} + 4L^2\Big(\mathbb{E}\|\bar{\theta}_\ell - \theta_\ell^{(n)}\|^2 + \mathbb{E}\|\mathcal{W}_{\bar{\theta}_\ell} - \mathcal{W}_\ell^{(n)}\|^2 \Big) \\
&\qquad + 8\zeta^2 + \frac{8L^2}{N}\sum_{j=1}^{N}\Big(\mathbb{E}\|\bar{\theta}_\ell - \theta_\ell^{(j)}\|^2 + \mathbb{E}\|\mathcal{W}_{\bar{\theta}_\ell} - \mathcal{W}_\ell^{(j)}\|^2 \Big)
\end{aligned} \tag{9}$$

Average over $n \in [N]$, we have:

$$\frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\Big\| G_\ell^{(n)} - \bar{G}_\ell \Big\|^2 \leq \frac{2\sigma^2}{b} + 8\zeta^2 + \frac{12L^2}{N}\sum_{n=1}^{N}\Big(\mathbb{E}\|\bar{\theta}_\ell - \theta_\ell^{(n)}\|^2 + \mathbb{E}\|\mathcal{W}_{\bar{\theta}_\ell} - \mathcal{W}_\ell^{(n)}\|^2 \Big)$$

So we have:

$$\begin{aligned}
\sum_{n=1}^{N} \|\theta_k^{(n)} - \bar{\theta}_k\|^2 &\leq r_\theta \sum_{\ell=\tilde{k}_r}^{k-1} \eta^2 \sum_{n=1}^{N} \|G_\ell^{(n)} - \bar{G}_\ell\|^2 \\
&\leq N r_\theta \sum_{\ell=\tilde{k}_r}^{k-1} \eta^2 \Big(\frac{2\sigma^2}{b} + 8\zeta^2 + \frac{12L^2}{N}\sum_{n=1}^{N}\Big(\mathbb{E}\|\bar{\theta}_\ell - \theta_\ell^{(n)}\|^2 + \mathbb{E}\|\mathcal{W}_{\bar{\theta}_\ell} - \mathcal{W}_\ell^{(n)}\|^2 \Big) \Big)
\end{aligned}$$

Then we combine with Lemma A.4 to get:

$$\begin{aligned}
&\sum_{n=1}^{N} \|\theta_k^{(n)} - \bar{\theta}_k\|^2 \\
&\leq N r_\theta \sum_{\ell=\tilde{k}_r}^{k-1} \eta^2 \bigg(\frac{2\sigma^2}{b} + 8\zeta^2 + \frac{12L^2}{\lambda}\Big(\frac{\gamma\sigma^2}{N} + 12Lr_{\mathcal{W}}\sigma^2\gamma^2 + 12Lr_{\mathcal{W}}^2\gamma^2\bar{\zeta}^2\Big) + \frac{12L^2}{N}\sum_{n=1}^{N}\Big(\mathbb{E}\|\bar{\theta}_\ell - \theta_\ell^{(n)}\|^2\Big)\bigg) \\
&\leq N r_\theta^2 \eta^2 \bigg(\frac{2\sigma^2}{b} + 8\zeta^2 + \frac{12L^2}{\lambda}\Big(\frac{\gamma\sigma^2}{bN} + 12Lr_{\mathcal{W}}\sigma^2\gamma^2 + 12Lr_{\mathcal{W}}^2\gamma^2\bar{\zeta}^2\Big)\bigg) + 12L^2 r_\theta \eta^2 \sum_{\ell=\tilde{k}_r}^{k-1}\sum_{n=1}^{N} \mathbb{E}\|\bar{\theta}_\ell - \theta_\ell^{(n)}\|^2
\end{aligned}$$

Next, we sum over k, we have:

$$(1 - 12L^2 r_\theta^2\eta^2) \sum_{k=\tilde{k}_r}^{\tilde{k}_r + r_\theta - 1} \sum_{n=1}^{N} \mathbb{E}\|\bar{\theta}_k - \theta_k^{(n)}\|^2 \leq N r_\theta^3 \eta^2 \bigg(\frac{2\sigma^2}{b} + 8\zeta^2 + \frac{12L^2}{\lambda}\Big(\frac{\gamma\sigma^2}{bN} + 12Lr_{\mathcal{W}}\sigma^2\gamma^2 + 12Lr_{\mathcal{W}}^2\gamma^2\bar{\zeta}^2\Big)\bigg)$$

This completes the proof. □

A.2. Descent Lemma

Lemma A.6. *Suppose $\eta\eta_g \leq \frac{1}{2IL}$ For $t \in [T]$, the iterates generated satisfy:*

$$\begin{aligned}
\mathbb{E}[h(\bar{\theta}_{k_{HN}+1})] \leq \mathbb{E}[h(\bar{\theta}_{k_{HN}})] - \frac{\eta}{2}\mathbb{E}\|\nabla h(\bar{\theta}_{k_{HN}})\|^2 - \frac{\eta}{4}\mathbb{E}\|\mathbb{E}_\xi[\bar{G}_{k_{HN}}]\|^2 \\
+ \frac{L^2\eta}{2N}\sum_{n=1}^{N}\Big(\mathbb{E}\|\bar{\theta}_{k_{HN}} - \theta_{k_{HN}}^{(n)}\|^2 + \mathbb{E}\|\mathcal{W}_{\bar{\theta}_{k_{HN}}} - \bar{\mathcal{W}}_{(r_{HN}+1)k_{HN}}\|^2\Big) + \frac{\eta^2 L\sigma^2}{2bN}
\end{aligned}$$

where the expectation is w.r.t the stochasticity of the algorithm.

Proof. Using the smoothness of h we have:

$$\begin{aligned}
\mathbb{E}[h(\bar{\theta}_{k_{HN}+1})] &\leq \mathbb{E}[h(\bar{\theta}_{k_{HN}})] + \mathbb{E}\langle \nabla h(\bar{\theta}_{k_{HN}}), \bar{\theta}_{k_{HN}+1} - \bar{\theta}_{k_{HN}}\rangle + \frac{L}{2}\mathbb{E}\|\bar{\theta}_{k_{HN}+1} - \bar{\theta}_{k_{HN}}\|^2 \\
&\overset{(a)}{=} \mathbb{E}[h(\bar{\theta}_{k_{HN}})] - \eta\mathbb{E}\langle \nabla h(\bar{\theta}_{k_{HN}}), \mathbb{E}_\xi[\bar{G}_{k_{HN}}]\rangle + \frac{\eta^2 L}{2}\mathbb{E}\|\mathbb{E}_\xi[\bar{G}_{k_{HN}}]\|^2 + \frac{\eta^2 L\sigma^2}{2bN} \\
&\overset{(b)}{=} \mathbb{E}[h(\bar{\theta}_{k_{HN}})] - \frac{\eta}{2}\mathbb{E}\|\nabla h(\bar{\theta}_{k_{HN}})\|^2 + \frac{\eta}{2}\mathbb{E}\|\nabla h(\bar{\theta}_{k_{HN}}) - \mathbb{E}_\xi[\bar{G}_{k_{HN}}]\|^2 \\
&\quad - \Big(\frac{\eta}{2} - \frac{\eta^2 L}{2}\Big)\mathbb{E}\|\mathbb{E}_\xi[\bar{G}_{k_{HN}}]\|^2 + \frac{\eta^2 L\sigma^2}{2bN} \\
&\overset{(c)}{\leq} \mathbb{E}[h(\bar{\theta}_{k_{HN}})] - \frac{\eta}{2}\mathbb{E}\|\nabla h(\bar{\theta}_{k_{HN}})\|^2 - \frac{\eta}{4}\mathbb{E}\|\mathbb{E}_\xi[\bar{G}_{k_{HN}}]\|^2 \\
&\quad + \frac{L^2\eta}{2N}\sum_{n=1}^{N}\Big(\mathbb{E}\|\bar{\theta}_{k_{HN}} - \theta_{k_{HN}}^{(n)}\|^2 + \mathbb{E}\|\mathcal{W}_{\bar{\theta}_{k_{HN}}} - \bar{\mathcal{W}}_{(r_{HN}+1)k_{HN}}\|^2\Big) + \frac{\eta^2 L\sigma^2}{2bN}
\end{aligned}$$

where equality (a) follows from the update step to the HN in Algorithm 1; (b) uses $\langle a, b\rangle = \frac{1}{2}[\|a\|^2 + \|b\|^2 - \|a-b\|^2]$; (c) follows the condition that $\eta_k \leq 1/(2L)$ and the smoothness of $h(\theta)$. This completes the proof. □

A.3. Proof of Convergence Theorem

We are ready to prove the main theorem 3.1 in this subsection.

Theorem A.7. *Suppose we the upper level learning rate η and the lower level learning rate γ as:*

$$\eta = \min\bigg\{\frac{1}{4Lr_\theta}, \Big(\frac{2bN\Delta_\theta}{K_{HN}L\sigma^2}\Big)^{1/2}\bigg\}, \quad \gamma = \min\{\frac{1}{4L}, \Big(\frac{\lambda bN}{K_{HN}L^2\sigma^2}\Big)^{1/2}\}$$

then we have:

$$\frac{1}{K_{HN}} \sum_{k_{HN}=0}^{K_{HN}-1} \mathbb{E}\|\nabla h(\bar{\theta}_{k_{HN}})\|^2 \leq \left(\frac{2L\sigma^2\Delta_\theta}{bNK_{HN}}\right)^{1/2} + \left(\frac{L^2\sigma^2}{\lambda bNK_{HN}}\right)^{1/2}$$

where b is the mini-batch size, N is the number of devices

Proof. Combine Lemma A.6, and Lemma A.4 to have:

$$\begin{aligned}\mathbb{E}[h(\bar{\theta}_{k_{HN}+1})] \leq \mathbb{E}[h(\bar{\theta}_{k_{HN}})] &- \frac{\eta}{2}\mathbb{E}\|\nabla h(\bar{\theta}_{k_{HN}})\|^2 - \frac{\eta}{4}\mathbb{E}\|\mathbb{E}_\xi[\bar{G}_{k_{HN}}]\|^2 \\ &+ \frac{L^2\eta}{2N}\sum_{n=1}^{N}\mathbb{E}\|\bar{\theta}_{k_{HN}} - \theta_{k_{HN}}^{(n)}\|^2 + \frac{L^2\eta}{2}(1-\lambda\gamma)^{r_{HN}}\Delta_{\mathcal{W}} \\ &+ \frac{L^2\eta}{2\lambda}\left(\frac{\gamma\sigma^2}{bN} + 12Lr_{\mathcal{W}}\sigma^2\gamma^2 + 12Lr_{\mathcal{W}}^2\gamma^2\bar{\zeta}^2\right) + \frac{\eta^2L\sigma^2}{2bN}\end{aligned}$$

Next, we sum over k_{HN}, and denote $\Delta_\theta = h(\theta_0)$, we have:

$$\begin{aligned}\frac{\eta}{2}\sum_{k_{HN}=0}^{K_{HN}-1}\mathbb{E}\|\nabla h(\bar{\theta}_{k_{HN}})\|^2 \leq \Delta_\theta &+ \frac{L^2\eta}{2N}\sum_{k_{HN}=0}^{K_{HN}-1}\sum_{n=1}^{N}\mathbb{E}\|\bar{\theta}_{k_{HN}} - \theta_{k_{HN}}^{(n)}\|^2 + \frac{K_{HN}L^2\eta}{2}(1-\lambda\gamma)^{r_{HN}}\Delta_{\mathcal{W}} \\ &+ \frac{K_{HN}L^2\eta}{2\lambda}\left(\frac{\gamma\sigma^2}{bN} + 12Lr_{\mathcal{W}}\sigma^2\gamma^2 + 12Lr_{\mathcal{W}}^2\gamma^2\bar{\zeta}^2\right) + \frac{K_{HN}\eta^2L\sigma^2}{2bN}\end{aligned}$$

Suppose we choose $\eta < \frac{1}{4Lr_\theta}$. By Lemma A.5, we have:

$$\sum_{k=\tilde{k}_r}^{\tilde{k}_r+r_\theta-1}\sum_{n=1}^{N}\mathbb{E}\|\bar{\theta}_k - \theta_k^{(n)}\|^2 \leq 4Nr_\theta^3\eta^2\left(\frac{2\sigma^2}{b} + 8\zeta^2 + \frac{12L^2}{\lambda}\left(\frac{\gamma\sigma^2}{bN} + 12Lr_{\mathcal{W}}\sigma^2\gamma^2 + 12Lr_{\mathcal{W}}^2\gamma^2\bar{\zeta}^2\right)\right)$$

So we have:

$$\begin{aligned}\frac{\eta}{2}\sum_{k_{HN}=0}^{K_{HN}-1}\mathbb{E}\|\nabla h(\bar{\theta}_{k_{HN}})\|^2 \leq \Delta_\theta &+ 2K_{HN}L^2r_\theta^2\eta^3\left(\frac{2\sigma^2}{b} + 8\zeta^2 + \frac{12L^2}{\lambda}\left(\frac{\gamma\sigma^2}{bN} + 12Lr_{\mathcal{W}}\sigma^2\gamma^2 + 12Lr_{\mathcal{W}}^2\gamma^2\bar{\zeta}^2\right)\right) \\ &+ \frac{K_{HN}L^2\eta}{2}(1-\lambda\gamma)^{r_{HN}}\Delta_{\mathcal{W}} \\ &+ \frac{K_{HN}L^2\eta}{2\lambda}\left(\frac{\gamma\sigma^2}{bN} + 12Lr_{\mathcal{W}}\sigma^2\gamma^2 + 12Lr_{\mathcal{W}}^2\gamma^2\bar{\zeta}^2\right) + \frac{K_{HN}\eta^2L\sigma^2}{2bN}\end{aligned}$$

Multiply $\frac{2}{\eta K_{HN}}$ on both sides, we have:

$$\begin{aligned}\frac{1}{K_{HN}}\sum_{k_{HN}=0}^{K_{HN}-1}\mathbb{E}\|\nabla h(\bar{\theta}_{k_{HN}})\|^2 \leq \frac{2\Delta_\theta}{\eta K_{HN}} &+ 4L^2r_\theta^2\eta^2\left(\frac{2\sigma^2}{b} + 8\zeta^2 + \frac{12L^2}{\lambda}\left(\frac{\gamma\sigma^2}{bN} + 12Lr_{\mathcal{W}}\sigma^2\gamma^2 + 12Lr_{\mathcal{W}}^2\gamma^2\bar{\zeta}^2\right)\right) \\ &+ L^2(1-\lambda\gamma)^{r_{HN}}\Delta_{\mathcal{W}} \\ &+ \frac{L^2}{\lambda}\left(\frac{\gamma\sigma^2}{bN} + 12Lr_{\mathcal{W}}\sigma^2\gamma^2 + 12Lr_{\mathcal{W}}^2\gamma^2\bar{\zeta}^2\right) + \frac{\eta L\sigma^2}{bN}\end{aligned}$$

By ignoring the higher order terms, we have:

$$\frac{1}{K_{HN}}\sum_{k_{HN}=0}^{K_{HN}-1}\mathbb{E}\|\nabla h(\bar{\theta}_{k_{HN}})\|^2 \leq \frac{2\Delta_\theta}{\eta K_{HN}} + \frac{\gamma L^2\sigma^2}{\lambda bN} + \frac{\eta L\sigma^2}{bN}$$

By choosing:

$$\eta = \min\left\{\frac{1}{4Lr_\theta}, \left(\frac{2bN\Delta_\theta}{K_{HN}L\sigma^2}\right)^{1/2}\right\}, \quad \gamma = \min\{\frac{1}{4L}, \left(\frac{\lambda bN}{K_{HN}L^2\sigma^2}\right)^{1/2}\}$$

then we have:

$$\frac{1}{K_{HN}}\sum_{k_{HN}=0}^{K_{HN}-1}\mathbb{E}\|\nabla h(\bar{\theta}_{k_{HN}})\|^2 \leq \left(\frac{2L\sigma^2\Delta_\theta}{bNK_{HN}}\right)^{1/2} + \left(\frac{L^2\sigma^2}{\lambda bNK_{HN}}\right)^{1/2}$$

this completes the proof. $\square$

Inputs $n = 0, \cdots, N$,
Embedding(N, 32× L), Resize to (L,32)
GRU(32, 64), LayerNorm, ReLU FC$_l$(64,C_l), LayerNorm, $l = 1, \cdots, L$
Outputs $\bar{\mathbf{a}}_l$, $l = 1, \cdots, L$

Table 5. The architecture of the embedding and the hypernetwork used in our method.

B. Experimental Settings

In Tab. 6, we provide the details choices of p_s and p_d^n. When training the base model, we use SGD as the local optimizer with a start learning rate of 0.1, momentum of 0.9, and weight decay of 0.0001. The learning rate is decayed to 0.01 and 0.001 at epochs 50 and 100, respectively. We set the mini-batchsize on each device to be $\lfloor 256/N \rfloor$, where $\lfloor \cdot \rfloor$ is the floor function. For FedOSP, we also use the hypernetwork for pruning, and the pruning process is conducted on the subset $D_n^{\mathbf{a}}$. We train the hypernetwork for FedOSP for 200 epochs with ADAM and a constant learning rate of 0.001. With these settings, the additional costs for training the hypernetwork for FedOSP and other methods will be similar. During the finetuning process, we use exactly the same setting as the base model training process.

We assume the local data distribution on each device is similar, and we split the test dataset to each device according to this assumption. In [48], for each class, they sample a vector $v \sim \text{Dir}(\alpha)$, where r is a N (the number of devices) dimensional vector which lies in a simplex ($\sum_{n=1}^{N} v_n = 1$). For each class, we modify this vector by adding random noise:

$$\bar{v} = v + r, \; r \sim \text{Uniform}(-\beta, \beta). \tag{10}$$

To satisfy the simplex requirement, we produce the final by using $\hat{v} = \frac{\bar{v}}{\sum_{n=1}^{N} \bar{v}_n}$. In experiments to satisfy our assumption, we let $\beta = \frac{1}{10N}$. We then assign test samples to each device based on $\hat{v}$.

C. Details of the Hypernetwork and Embedding

The detailed architecture of the embedding and the hypernetwork is shown in Tab. 5. GRU [4] is used to capture inter-layer relationships and fully connected layers are used to capture inter-channel interactions.

After we have the outputs $\bar{\mathbf{a}}_l$ from the hypernetwork, we calculate the binary vectors $\mathbf{a}_l$ by using the following equations:

$$\mathbf{a}_l = \text{round}(\text{sigmoid}((\bar{\mathbf{a}}_l + g + c)/\tau)), \tag{11}$$

where $g \sim \text{Gumbel}(0, 1)$, and Gumbel is the Gumbel distribution, and c is a constant to make the pruning starts with

Dataset	Architecture	p_d^n	p_s
CIFAR-10	ResNet-56	0.50	0.80
CIFAR-100	ResNet-18	0.50/0.30	0.80/0.60
	ResNet-34	0.50/0.30	0.80/0.60
	MobileNet-V2	0.50	0.80
TinyImageNet	ResNet-18	0.50	0.80
	ResNet-34	0.50	0.80

Table 6. Choice of p_r and p_p.

the whole network. In practice, we set $c = 3.0$ for all experiments. τ is the temperature parameter, which is set to 0.4 for all experiments.

D. Ablation Study

Settings	Base Acc	Δ-Acc	Acc	↓ FLOPs (D)	↓ FLOPs (S)
FedILP	66.57%	-0.20%	66.37%	50%	50%
DWNP (OS)		+1.02%	67.59%	50%	50%
DWNP w/o Eq. 7		+1.43%	68.00%	50%	50%
DWNP		**+1.74%**	**68.31%**	50%	20%

Table 7. Results of CIFAR-100 with different settings.

In Tab. 7, we add more settings to study the effects of the design choices. 'DWNP (OS)' corresponds to one-shot pruning for server and device sub-networks. 'DWNP w/o Eq. 7' corresponds to not resolving the conflicts between server and device sub-networks. The one-shot setting results in 0.72% performance loss, which suggests that co-training indeed produces a better final model, as we discussed in section 3.6. The difference between 'DWNP w/o Eq. 7' and 'DWNP' is not large, but the cost of adding Eq. 7 is minimal. Thus, the performance gain is nearly free, and it also suggests that restricting the search space of the device-wise sub-networks is helpful.

E. Control the Relationship between the Server-side and Device-side Sub-networks

To better describe the relationship between the server-side sub-network $\mathbf{a^0}$ and device-side sub-networks $\mathbf{a^n}$, we explicitly require that if a channel is pruned by the server-side sub-network, the corresponding device-side sub-networks should not update the corresponding position. This is achieved by stopping gradients if the condition is met:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^n[i]} = 0, \text{ if } i \in \text{Ind}(a^0), \tag{12}$$

where we define the index set $\text{Ind}(a^0)$ to be $\text{Ind}(a^0) = \{i \mid \mathbf{a^0}[i] = 0\}$, which is used to represent the index of pruned positions given the server-side sub-network. This constraint is dynamically changed during the learning process of sub-networks, but it will become stable in the middle to the late training stage. The benefits of this constraint are two-fold. Firstly, it implicitly embeds the relationship

Method	Architectures	Time (sec/img)	↓ Time	↓ FLOPs (D)	↓ FLOPs (S)
Original	ResNet-18	0.0559	-	-	-
Filter Pruning		0.0348	37.7%	50%	50%
FedOSP		0.0336	39.9%	50%	50%
FedILP		0.0345	38.3%	50%	50%
DWNP		0.0340	39.2%	50%	20%

Table 8. Inference Time Comparison on TinyImageNet.

Architecture	Method	Base Top-1 Acc	Δ Top-1 Acc	Acc	↓ FLOPs (D)	↓ FLOPs (S)
ResNet-18	Filter Pruning	73.52%	-1.28%	72.24%	50%	50%
	FedOSP		-0.32%	73.20%	50%	50%
	FedILP		-0.04%	73.48%	50%	50%
	DWNP		**+1.42%**	74.94%	50%	20%

Table 9. Comparison results on ImageNet-100 with ResNet-18.

between the server-side sub-network and device-side sub-networks. Secondly, it reduces the potential search space for device-side sub-networks, which is beneficial for learning them.

F. ImageNet-100 Results

We use the ImageNet-100 subset to further evaluate our method following the partition shown in [5]. For ImageNet-100, We follow the same training setting of other datasets described in section 4.1, except that we reduced the training and finetuning epochs to 80 to save computational costs. We use 8 clients with $\alpha = 0.5$ on ImageNet-100. The comparison result is shown in Tab. 9. DWNP still consistently outperforms other baselines like other datasets. This experiment further demonstrates that our method can be easily scaled up to larger (higher resolution/more samples) datasets.

G. Inference Time Comparasion

We measure the inference time on the CPU with ResNet-18 on tiny-ImageNet in Tab. 8. The result is obtained by averaging inference time from 1000 input samples. For DWNP, the time is further averaged from 10 device-side sub-networks, and the range of inference time for DWNP is $0.0336 \sim 0.0351$ (sec/img). From the table, we can see that different methods have comparable acceleration rates giving similar FLOPs on each device.