How Few Davids Improve One Goliath: Federated Learning in Resource-Skewed Edge Computing Environments

Jiayun Zhang
University of California, San Diego
jiz069@ucsd.edu

Shuheng Li
University of California, San Diego
shl060@ucsd.edu

Haiyu Huang
University of California, Los Angeles
haiyu@g.ucla.edu

Zihan Wang
University of California, San Diego
ziw224@ucsd.edu

Xiaohan Fu
University of California, San Diego
xhfu@ucsd.edu

Dezhi Hong*
Amazon
hondezhi@amazon.com

Rajesh K. Gupta
University of California, San Diego
rgupta@ucsd.edu

Jingbo Shang
University of California, San Diego
jshang@ucsd.edu

ABSTRACT

Real-world deployment of federated learning requires orchestrating clients with widely varied compute resources, from *strong* enterprise-grade devices in data centers to *weak* mobile and Web-of-Things devices. Prior works have attempted to downscale large models for weak devices and aggregate shared parts among heterogeneous models. A typical architectural assumption is that there are equally many strong and weak devices. In reality, however, we often encounter *resource skew* where a few (1 or 2) strong devices hold substantial data resources, alongside many weak devices. This poses challenges—the unshared portion of the large model rarely receives updates or gains benefits from weak collaborators.

We aim to facilitate reciprocal benefits between strong and weak devices in resource-skewed environments. We propose RecipFL[1], a novel framework featuring a server-side graph hypernetwork. This hypernetwork is trained to produce parameters for personalized client models adapted to device capacity and unique data distribution. It effectively generalizes knowledge about parameters across different model architectures by encoding computational graphs. Notably, RecipFL is agnostic to model scaling strategies and supports collaboration among arbitrary neural networks. We establish the generalization bound of RecipFL through theoretical analysis and conduct extensive experiments with various model architectures. Results show that RecipFL improves accuracy by 4.5% and 7.4% for strong and weak devices respectively, incentivizing both devices to actively engage in federated learning.

CCS CONCEPTS

• **Computing methodologies** → **Distributed artificial intelligence**; **Machine learning**; • **Human-centered computing** → **Ubiquitous and mobile computing**.

*Work unrelated to Amazon.
[1]Code is available on Github: https://github.com/jiayunz/RecipFL.

WWW '24, May 13–17, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0171-9/24/05.
https://doi.org/10.1145/3589334.3645544

KEYWORDS

federated learning, edge computing, system heterogeneity

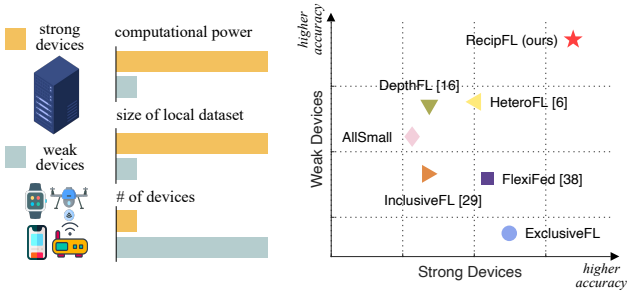
ACM Reference Format:
Jiayun Zhang, Shuheng Li, Haiyu Huang, Zihan Wang, Xiaohan Fu, Dezhi Hong, Rajesh K. Gupta, and Jingbo Shang. 2024. How Few Davids Improve One Goliath: Federated Learning in Resource-Skewed Edge Computing Environments. In *Proceedings of the ACM Web Conference 2024 (WWW '24), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3589334.3645544

1 INTRODUCTION

The growing demand for data privacy has catalyzed the rise of federated learning [30] as a privacy-preserving distributed learning paradigm. The real-world deployment of federated learning needs to deal with heterogeneous edge computing environments [14, 21, 35]. Typically, a few devices, often owned by large enterprises, can be powerful enough to afford large models, while the vast majority are 'weak' devices that can only host small models, such as mobile and Web-of-Things (WoT) devices owned by individuals. Universally deploying homogeneous small models as required by traditional methods [18, 19] not only wastes available compute resources but also compromises performance. Ideally, we need models scaled to fit varying device capacities and perform effective model aggregation.

To support collaboration among heterogeneous models, prior methods [6, 16, 23, 29, 31] downscale the large model for weak devices and perform aggregation on common components. These works typically assume there are equally many strong and weak devices [16, 29, 38]. However, in reality, we often see a skewed computing environment where a small number of strong devices operated by enterprises are accompanied by a large number of user-owned weak devices. For example, a smartwatch company wants to develop an activity recognition system. The company trains a large model using a vast dataset gathered from controlled environments, while its smartwatch users join via federated learning to train small models using personal data in the wild. Although small models are expected to benefit from the large model [2, 10], their contribution to the large model is dubious given their limited capability.

In light of this gap, we explore a new research question: *Can strong devices benefit from weak devices in resource-skewed environments?* We consider an extreme scenario where limited (1 or 2)



(a) Resource Skew (b) Performance Summary

Figure 1: Illustration of problem setting and the performance of prior methods in resource-skewed environments.

strong devices and numerous weak devices engage in the learning, as depicted in Figure 1(a). In this scenario, the learning system heavily leans on weak devices, leaving the unshared portion of the large model rarely being updated or deriving benefits from others. This presents a significant challenge in improving strong devices.

Existing approaches employ either width-scaling to prune channels or neurons in each layer of the large model [6, 23, 31], or depth-scaling for layer-wise pruning [16, 29]. They rely on weight-averaging aggregation [25, 30] to update shared layers. However, it can be destructive when layers or neurons in small models are ill-aligned with those in large models. For example, if the first block of ResNet [9] operates as an independent model for a complete vision recognition process, its layers function differently compared to their counterparts within an entire ResNet. When a few full ResNets are aggregated with many of their smaller versions (i.e., first blocks only), the first block may only extract shallow vision features, leading to performance decline. Even facilitated with knowledge distillation [16], improvements are not guaranteed if knowledge is transferred from numerous small models biased by non-IID data, as corroborated by our experiment results.

To effectively align and aggregate heterogeneous models, we propose RecipFL, a novel federated learning framework that empowers the server with a graph hypernetwork tasked with producing personalized model parameters for clients. Clients retain the flexibility to adapt the model to their capacities, through pruning or architectural changes. The server transforms client models into directed acyclic graphs to delineate their computation flow among layers. Figure 2 presents the overview of RecipFL. Unlike traditional weight-averaging aggregation, which requires layers to have uniform operations, sizes, and computational flows, graph hypernetwork supports collaboration among arbitrary model architectures. This is achieved by encoding the computational graphs of client models with a graph neural network (GNN) [27, 33] and decoding parameters with multi-layer perceptrons (MLPs). It captures shared patterns among model architectures, such as residual block and convolution patterns, and generalizes knowledge across them. The hypernetwork is trained using feedback (i.e., updated weights) from clients during federated learning. The computations of hypernetwork are executed by the server and therefore do not add extra communication or computation overhead to edge devices. We further augment weak devices by distilling knowledge from large models to smaller ones on strong devices.

We theoretically analyze the generalization bound of RecipFL and empirically evaluate RecipFL framework across four datasets for image classification and natural language inference. We simulate non-IID client distributions and evaluate personalized client models on their own test data as real-world WoT and mobile computing applications typically require. The results show RecipFL outperforms state-of-the-art methods across different scaling strategies and various model architectures with significant margins. Notably, RecipFL yields improvements for both strong and weak devices, demonstrating that even devices with limited computational resources can contribute meaningfully to the learning system, thereby providing incentives to both devices to participate in federated learning.

Our contributions are summarized as follows:

- We address a new research question in federated learning: *Can strong devices benefit from weak devices in resource-skewed environments?* We show the existing approaches do not guarantee improvement for both types of devices.
- We propose a novel framework RecipFL to effectively generate weights for heterogeneous client models based on graph hypernetwork, compatible with arbitrary model scaling strategies.
- We establish the generalization bound of RecipFL through theoretical analysis and validate its performance through extensive experiments. RecipFL outperforms various state-of-the-art methods with significant margins and demonstrates that weak devices can also contribute effectively to the learning of strong devices.

2 PRELIMINARIES

2.1 Problem Definition

We aim to build a federated learning system with M clients that allows the clients to have customized model architectures $\{\mathcal{G}_m | m \in [M]\}$ that fit their specific running capabilities. Within the M clients, there are a few (e.g., 1 or 2) strong devices that have enough running capacity to hold large models and the rest are weak devices having limited computing power. Denote the training set on client m as $\mathbf{D}_m = \left\{\left(x_i^{(m)}, y_i^{(m)}\right)\right\}_{i=1}^{N_m}$, where $x_i^{(m)}$ is the input data and $y_i^{(m)}$ is the label, and the data distribution of client m as $\mathcal{P}_m$. Denote ℓ as the loss function. The goal is to learn a personalized model $f_m(\cdot;\theta_m)$ for every client m that works on its own data distribution:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{(x,y)\sim\mathcal{P}_m} \ell(f_m(x;\theta_m), y), \tag{1}$$

where $\boldsymbol{\theta}$ is the set of client model weights: $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_M\}$.

2.2 Resource-Skewed Computing Environments

Existing works often assume the strong and weak devices are equally distributed [16, 29, 38]. Among these, a tangential sensitivity evaluation [29] indicates that when strong devices are the minority, the large models deployed on them converge slowly and the performance is even worse than training them without the participation of weak devices. Yet, there is a lack of analysis or solutions for this resource skew issue.

To understand how existing methods perform in skewed computing environments, we summarize the results from Section 5 by averaging the accuracies across all datasets for strong and weak

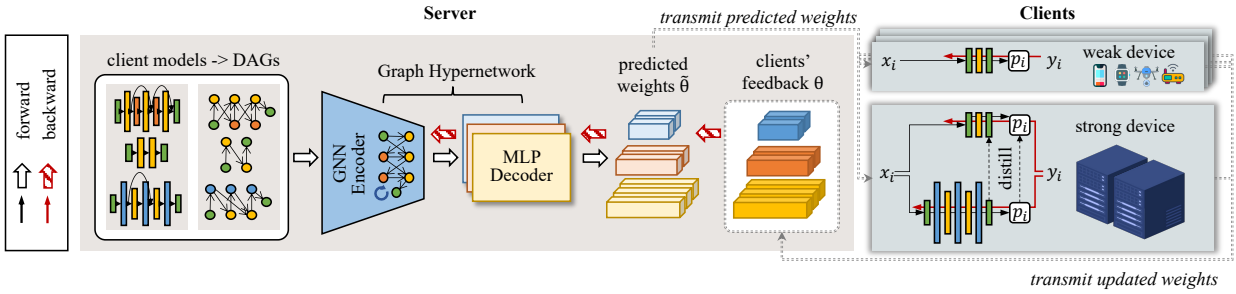


Figure 2: Overview of RecipFL framework. The server transforms client models into directed acyclic graphs (DAGs) to represent the computation flow among operations and trains a graph hypernetwork to generate weights for customized client models.

devices respectively. Our experiments use a majority of weak devices (e.g., 5, 20, 50, 500) and 1 or 2 strong devices, allocating 50% of data to strong devices and the rest to weak devices under Dirichlet distributions. We craft two naive baselines: AllSmall, which trains small models on all devices via federated learning, and ExclusiveFL, which trains large models on strong devices and small models on weak devices, with weight aggregation carried out separately within each group. In addition, we include comparisons with existing federated methods for heterogeneous models [6, 16, 29, 38]. We evaluate client models on their own test data sampled from clients' data distributions. The summary is presented in Figure 1(b), from which we draw the following observations:

- *Small models are insufficient for strong devices:* By comparing AllSmall and ExclusiveFL on strong devices, we see that training small models with collaboration from weak devices yields lower accuracy compared to training large models independently.
- *Weak devices benefit from collaboration with strong devices:* By comparing ExclusiveFL and other methods on weak devices, we observed that all other methods show higher accuracy than training small models independently.
- *Strong devices derive minimal benefits from weak devices with existing methods:* For strong devices, we see the accuracy of existing methods is generally lower than ExclusiveFL.
- *Existing methods could enhance the performance of one type of model but struggle to improve both.* For example, DepthFL achieves high accuracy on weak devices but does not perform well on strong devices. FlexiFed achieves higher accuracy on strong devices than DepthFL but shows less improvement on weak devices.

Recognizing these limitations, we aim to enable mutual benefits between both types of devices in resource-skewed environments.

3 OUR RECIPFL FRAMEWORK

RecipFL employs a graph hypernetwork at the server that learns from client feedback during federated learning. This hypernetwork encodes clients' computational graphs, enabling it to generalize knowledge across different model architectures and produce personalized client parameters. Algorithm 1 outlines the pseudo code.

3.1 Federated Training

In each training round, the server initiates the process by randomly selecting a subset of clients, denoted as $\mathbf{S}_t$, to conduct local updates. The server utilizes the graph hypernetwork to produce

Algorithm 1: RecipFL Framework

Input : Communication rounds T, number of selected clients per round $|\mathbf{S}_t|$, local training epochs E, client descriptors $\{a_m | m \in [M]\}$ and model architectures $\{\mathcal{G}_m | m \in [M]\}$.

Output : A graph hypernetwork that generates personalized model weights for heterogeneous client models.

Server executes:

for $t = 1, \ldots, T$ **do**
  Select a subset $\mathbf{S}_t$ of clients at random;
  for $m \in \mathbf{S}_t$ **do**
    $\tilde{\theta}_m \leftarrow \text{GHN}(\mathcal{G}_m, a_m; \phi)$;
    $\theta_m \leftarrow$ **ClientUpdate**$(m, \tilde{\theta}_m)$;
    $\Delta\theta_m \leftarrow \theta_m - \tilde{\theta}_m$;
  Update GHN: $\phi \leftarrow \phi - \eta_s \sum_{m \in \mathbf{S}_t} (\nabla_\phi \theta_m)^T \Delta\theta_m$;
return $\text{GHN}(\cdot; \phi)$;

ClientUpdate$(m, \tilde{\theta}_m)$:

$\theta_m \leftarrow \tilde{\theta}_m$;
for $e = 1, \ldots, E$ **do**
  Partition $\mathbf{D}_m$ into mini-batches $\bigcup_{i=1}^{j_m} B_i^{(m)}$;
  for $i = 1, \ldots, j_m$ **do**
    $\theta_m \leftarrow \theta_m - \eta_c \nabla_{\theta_m} \mathcal{L}_m(\theta_m; B_i^{(m)})$;
return θ_m to server ;

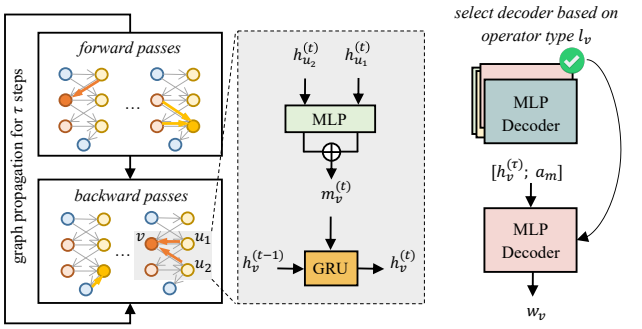
model weights $\{\tilde{\theta}_m | m \in \mathbf{S}_t\}$, sends the weights to selected clients and waits for their feedback. At the client side, the client performs local updates by training the client model f_m with its local dataset $\mathbf{D}_m$. The training objective at client m is to minimize the loss:

$$\underset{\theta}{\text{argmin}}\, \mathcal{L}_m(\theta_m) = \underset{\theta}{\text{argmin}} \frac{1}{N_m} \sum_{i=1}^{N_m} \ell\left(f_m\left(x_i^{(m)}; \theta_m\right), y_i^{(m)}\right), \quad (2)$$

where N_m is the number of samples in the local dataset $\mathbf{D}_m$. Let η_c be the learning rate for local training at the client. Starting with the initial value $\theta_m = \tilde{\theta}_m$, the client updates θ_m as follows:

$$\theta_m \leftarrow \theta_m - \eta_c \nabla_{\theta_m} \mathcal{L}_m(\theta_m). \quad (3)$$

After local training, the clients send the updated model weights back to the server. The server then calculates the change in local



(a) Encoding (b) Decoding

Figure 3: Graph hypernetwork architecture.

model parameters $\Delta\theta_m = \theta_m - \tilde{\theta}_m$, and uses the chain rule $\nabla_\phi \mathcal{L}_m = (\nabla_\phi \theta_m)^T \nabla_{\theta_m} \mathcal{L}_m$ to update the graph hypernetwork parameter ϕ:

$$\phi \leftarrow \phi - \eta_s \sum\nolimits_{m \in \mathrm{S}_t} (\nabla_\phi \theta_m)^T \Delta\theta_m, \tag{4}$$

where η_s is the learning rate for updating the graph hypernetwork. By adopting the graph hypernetwork, we modify Equation 1 and formulate the new learning objective as:

$$\mathrm{argmin}_\phi \hat{\mathcal{L}}(\phi, \boldsymbol{D}), \tag{5}$$

where $\hat{\mathcal{L}}(\phi, \boldsymbol{D}) = \frac{1}{M} \sum_{m=1}^{M} \mathcal{L}_m(\mathrm{GHN}(\mathcal{G}_m, a_m; \phi))$ is the average empirical loss on dataset $\boldsymbol{D} = \{\boldsymbol{D}_m\}_{m=1}^{M}$.

Importantly, the graph hypernetwork resides on the server, with all computations related to the graph hypernetwork executed solely by the server. This design ensures it does not impose additional communication or computational overhead on clients.

3.2 Weight Generation with Hypernetwork

Representation of target network architectures. We represent the computational graph of a neural network model as a directed acyclic graph, denoted as $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where the nodes $\mathcal{V}$ are the operators (e.g., convolution, pooling, linear layer, etc.) and the directed edges $\mathcal{E}$ describe the computation flow in the order of forward propagation among the operators. Conventional graph hypernetworks are inefficient in dealing with repeated similar local connection patterns in deep networks such as the ResNet blocks in ResNet-152. To enhance the ability to distinguish local connection patterns in target networks, we inform the graph hypernetwork about the model parameters of the target network at the current training round. To do so, the node features $\{h_v | v \in \mathcal{V}\} = \{[l_v, q_v] | v \in \mathcal{V}\}$ consist of two parts: (1) one-hot vectors l_v indicating the operations performed by the node, and (2) the current parameters q_v of the operators. A linear embedding layer transforms the one-hot vector l_v to a dense vector and a Transformer encoder [36] maps the variable-length node parameters q_v to a fixed-dimensional vector. The linear embedding layer and the Transformer encoder are learnable and are updated during training.

Graph hypernetwork architecture. As depicted in Figure 3, the graph hypernetwork consists of an encoding process that extracts features from node information and a decoding process that predicts weights for parametric operators according to the encoded features.

In the encoding phase, a graph neural network (GNN) [27, 33] is employed to conduct τ steps of graph propagation within $\mathcal{G}(\mathcal{V}, \mathcal{E})$ of the target network. During this graph propagation, the GNN topologically traverses the nodes in both forward and backward directions, iteratively conducting message passing and updating node features. For the t-th propagation step, the GNN first forward traverses nodes. Every node v receives messages from its incoming nodes and sends messages to its outgoing nodes. Denote the incoming nodes to node v as $\mathbf{IN}(v)$. The message function is modeled with an MLP shared among all the nodes. The message received by node v at step t is:

$$m_v^{(t)} = \sum\nolimits_{u \in \mathbf{IN}(v)} \mathrm{MLP}(h_u^{(t)}) \tag{6}$$

The node feature vector $h_v^{(t)}$ is then updated based on the aggregated message $m_v^{(t)}$ and the feature vector of node v at step $t-1$ using a Gated Recurrent Unit (GRU) cell [4]:

$$h_v^{(t)} = \mathrm{GRU}(h_v^{(t-1)}, m_v^{(t)}) \tag{7}$$

After traversing $\mathcal{G}(\mathcal{V}, \mathcal{E})$ in forward propagation, the GNN reverses the traversal direction and updates the node features again, i.e. receives messages from its incoming nodes along backward passes and sends to its outgoing nodes.

In the decoding phase, we use an individual MLP as the decoder for each type of parametric operator to generate parameters. To further support personalization, we introduce client descriptors $\{a_m | m \in [M]\}$ that describe the data characteristics of every client m. This descriptor is provided as input to the MLP decoder. Specifically, we use the class distribution of local training samples as the client descriptor. Alternatively, the client descriptor can simply be the client IDs, and in that case, a linear embedding layer can be used to transform them into client embeddings, enabling the learning of client features through training. Let $\mathrm{MLP}_l(\cdot)$ represent the decoder for the l-type operator. MLP_l operates on the concatenation of the node embedding and the client embedding, denoted as $[h_v^{(\tau)}, a_m]$, and generates parameters for the node. The resulting set of generated weights for the target network is:

$$w = \{w_v | v \in \mathcal{V}\} = \{\mathrm{MLP}_{l_v}([h_v^{(\tau)}, a_m]) | v \in \mathcal{V}\} \tag{8}$$

To handle different dimensionalities of layers within the same operator type, the outputs of the decoder are reshaped through tiling and concatenation to match the shape of the target layers following common practices in graph hypernetworks [17, 40].

3.3 Strong-to-Weak Device Knowledge Transfer

To further enhance the learning of small models, we leverage the computing resources on strong devices and employ regularizations to distill knowledge from large models to small ones.

For strong devices, we let the central graph hypernetwork generate weights for both small and large models. Denote the small and large model at the strong device m as f_m^S and f_m^L respectively and the corresponding model parameters are θ_m^S and θ_m^L. After training the large model f_m^L, we proceed to train the small model and distill knowledge from the large one. We introduce an additional cross-entropy loss term $CE(\cdot)$ to let the small model mimic the prediction probabilities of the large model. In addition, if the representations generated by the last hidden layers of the models have the same

dimension, we add a KL-divergence loss term $D_{KL}(\cdot)$ to align the feature spaces. Denote the softmax probability distributions of features generated by the last hidden layers of the small model and the large model as p_i^S and p_i^L respectively. The optimization objective for training small model f_m^S on strong device m is to minimize the following loss:

$$\mathcal{L}_m^S(\theta) = \frac{1}{n}\sum_{i=1}^{n}[CE(f_m^S(x_i;\theta_m^S), y_i) + CE(f_m^S(x_i;\theta_m^S), f_m^L(x_i;\theta_m^L)) + D_{KL}(p_i^L \| p_i^S)]. \tag{9}$$

After local training, the updated weights of both small and large models are sent to the server and used for the update of the graph hypernetwork. This knowledge transfer mechanism helps small models benefit from the insights learned by strong devices.

4 ANALYSIS ON GENERALIZATION BOUND

In this section, we establish the generalization bound of RecipFL.

Consider a training set on clients $\mathbf{D}_m = \left\{\left(x_i^{(m)}, y_i^{(m)}\right)\right\}_{i=1}^{N}$ for some natural number $N \geq 1$, i.e. we sample uniformly N training data from each data distribution $\mathcal{P}_m$ on client m for $m = 1, \cdots, M$.

Assume the loss function ℓ takes value in $[0, 1]$, or equivalently with rescaling, ℓ is bounded. Let d be the dimension of the hypernetwork parameter ϕ and assume $\phi \in [-R, R]^d$ for some large $R > 0$. Finally, assume the loss l is Lipschitz with respect to ϕ with Lipschitz constant $K > 0$, i.e. $|\ell\left(f_m\left(x; \mathrm{GHN}(\mathcal{G}_m, a_m; \phi)\right), y\right) - \ell\left(f_m\left(x; \mathrm{GHN}(\mathcal{G}_m, a_m; \phi')\right), y\right)| \leq K\|\phi - \phi'\|$ for all x, y and $m = 1, \ldots, M$. Here $\|\cdot\|$ denotes the Euclidean distance on $\mathbb{R}^d$. Define the expected loss as:

$$\mathcal{L}(\phi) = \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{(x,y)\sim\mathcal{P}_m}\ell(f_m(x; (\mathcal{G}_m, a_m; \phi)), y). \tag{10}$$

Theorem 4.1. *If the number of samples on each client satisfies*

$$N \geq \max\left\{\frac{4d}{M\epsilon^2}\log\left\lceil\frac{4RK\sqrt{d}}{\epsilon}\right\rceil + \frac{4}{M\epsilon^2}\log\frac{4}{\delta}, \frac{1}{\epsilon^2}\right\}, \tag{11}$$

then with probability at least $1 - \delta$ with respect to the probability distribution on $\boldsymbol{D} = \{\boldsymbol{D}_m\}_{m=1}^{M}$, $\mathcal{L}(\phi) < \hat{\mathcal{L}}(\phi, \boldsymbol{D}) + \epsilon$ for every ϕ.

The proof and more details are given in the Appendix. From Equation 11, we observe that the number of training samples N per device required for generalization is negatively related to the number of devices M, which suggests that introducing new weak devices to the system can help lower the threshold for generalization. Moreover, when there is a strong device possessing a large amount of data, it can also lower the threshold for weak devices. For example, if there is one strong device and M weak devices, we can regard the strong one as k virtual devices, which increases the total number of devices to $M + k$, and thereby lowers the threshold for the number of samples on weak devices. The only requirement is that the strong device then needs to take on k times more data samples than that is required for a weak device.

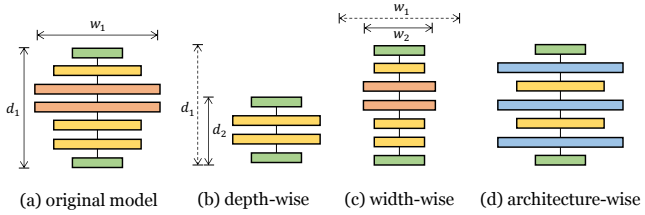
5 EXPERIMENTS

5.1 Experiment Setup

Configurations are summarized in Table 2. We provide details below.

Table 1: RecipFL is compatible with various ways of model scaling, showing more flexibility than existing solutions.

Scaling Strategy	HeteroFL	InclusiveFL	FlexiFed	DepthFL	RecipFL (ours)
depth-wise		✓	✓	✓	✓
width-wise	✓				✓
architecture-wise					✓



(a) original model (b) depth-wise (c) width-wise (d) architecture-wise

Figure 4: Illustration of model scaling strategies. The rectangle blocks represent the layers in neural networks. Different colors indicate different operations (e.g., convolution).

Datasets. We evaluate RecipFL on two fundamental categories of machine learning tasks: image classification with CIFAR-10 [20], CIFAR-100 [20], MNIST [22], and natural language inference with MNLI [39]. We simulate quantity skew where strong devices possess a dominant amount of data, as it often occurs in realistic resource-skewed environments. We allocate 50% of the entire dataset to the strong devices, while the weak devices evenly share the remaining half. This ensures the total amount of the data owned by weak devices is comparable to that owned by strong devices, making it possible for weak devices to contribute to the model enhancement of strong devices. Note that we conduct exploration studies in Section 5.5 to investigate the impact of data ratio by changing this configuration. To simulate non-IID client distributions, we follow the prior work [12, 16] and employ Dirichlet distribution $Dir(\alpha = 0.5)$ to sample the class distribution for every client. For the strong device, we assume it follows the universal distribution due to its substantial data volume.

Model architectures. To evaluate the robustness of our framework, we experiment with various popular neural network architectures. The large models include ResNet-18 [9], DenseNet-121 [13], LeNet-5 [22] and BERT [5]. Our framework is compatible with different ways of model scaling and we test all three scaling strategies shown in Figure 4. For depth-scaling, we follow [16] and regard the first block of ResNet-18 and DenseNet-121 as the small models. For width-scaling, we follow [6] and shrink the channels and hidden layers of the large model based on a scaling ratio. In order to achieve comparable model sizes with depth-scaling, we carefully set the scaling ratio for width-scaling by comparing the parameters in the depth-scaled models to those in the large models. In addition, we craft a smaller version of LeNet-5 which reserves the first block of LeNet-5 and scales the rest layers along the width. By doing this, we enable both depth- and width-wise aggregation for the comparison of existing methods. For architecture-wise scaling, We use DistilBERT [32] as the smaller version of BERT.

Enabling fine-tuning from pretrained models. RecipFL can support fine-tuning by inserting adapters [11], a small set of new parameters, and classification heads into pretrained models. During

Table 2: Federated learning configurations.

Dataset	# of devices		Data allocation		Large model	# of parameters			Pretrained?
	Strong	Weak	Strong	Weak		Original	Depth-scaled	Width-scaled	
CIFAR-10	1	5	50%	10%	ResNet-18	11M	450K	444K	✗
CIFAR-100	1	50	50%	1%	DenseNet-121	1M	258K	276K	✗
MNIST	2	500	25%	0.1%	LeNet-5	44K	5.6K (scaled in depth & width)		✗
MNLI	1	20	50%	2.5%	BERT	110M	67M (DistilBERT)		✓

Table 3: Experiment results (average accuracy and standard deviation). RecipFL consistently outperforms the compared methods across all datasets and model scaling strategies, benefiting both strong and weak devices.

Scaling	Method	CIFAR-10		CIFAR-100	
		Strong	Weak	Strong	Weak
Depth	AllSmall	64.34±2.14	68.50±3.42	17.86±2.56	25.56±3.11
	ExclusiveFL	84.85±1.85	59.11±4.22	32.21±3.81	19.22±2.07
	FlexiFed [38]	82.86±1.77	67.66±3.93	28.60±3.48	27.84±2.98
	InclusiveFL [29]	83.22±0.47	67.66±3.14	18.98±3.49	28.71±2.87
	DepthFL [16]	73.90±1.49	78.16±1.48	22.08±3.58	36.83±2.87
	RecipFL	**85.28±0.22**	**78.65±1.35**	**41.63±2.24**	**45.52±3.12**
Width	AllSmall	82.86±1.77	78.90±2.87	29.80±3.32	37.90±2.83
	ExclusiveFL	83.96±1.97	70.65±3.99	32.22±6.66	24.49±3.52
	HeteroFL [6]	84.76±1.19	77.93±2.92	26.51±2.70	39.05±2.82
	RecipFL	**85.06±0.13**	**82.88±1.29**	**43.64±2.84**	**42.00±3.88**

Method	MNIST		MNLI	
	Strong	Weak	Strong	Weak
AllSmall	91.73±3.94	77.05±7.47	73.47±0.52	82.13±2.89
ExclusiveFL	92.97±1.98	77.85±5.14	80.20±0.20	70.52±6.04
FlexiFed [38]	92.70±2.72	73.89±8.08	79.65±0.18	82.31±6.15
InclusiveFL [29]	84.77±3.12	75.72±6.27	79.87±0.30	81.17±4.31
DepthFL [16]	94.33±1.95	78.39±7.52	77.11±0.90	80.92±6.64
HeteroFL [6]	89.53±3.22	75.95±8.01	79.65±0.18	82.31±6.15
RecipFL	**97.07±1.87**	**86.36±6.60**	**82.78±0.57**	**83.37±4.72**

training, only the adapters and classification heads are updated and communicated between the server and clients, while the other layers are fixed at local. We initialize BERT[2] and DistilBERT[3] with pretrained weights provided by HuggingFace.

Compared methods. First, we construct two naive baselines based on the classical federated learning algorithm FedAvg [30]:

- **AllSmall**: All clients deploy the small models to compromise the smallest running capacity and conduct federated learning.
- **ExclusiveFL**: Clients with the same level of capacity are equipped with the same model, i.e., strong devices deploy large models while weak devices deploy small models. Each type of device performs weight aggregation exclusively.

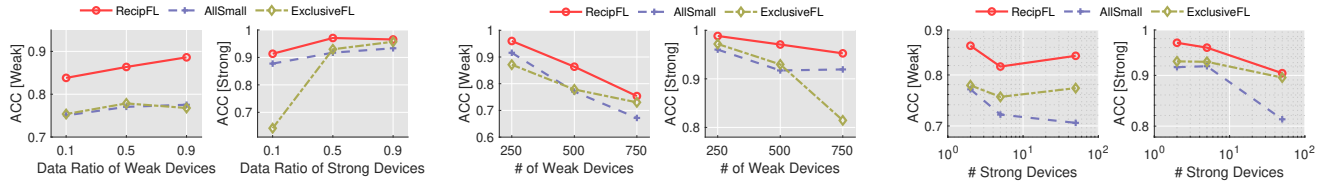
The performance of weak devices under AllSmall and that of strong devices under ExclusiveFL serve as reference points for assessing whether a method enhances the performance of weak or strong devices. We then compare RecipFL with state-of-the-art methods for federated learning with heterogeneous models:

- **HeteroFL** [6] adopts width-scaling where channels and hidden layers are scaled according to a fixed ratio. The global layer updates a subset of parameters correspondingly from scaled layers and all parameters from unscaled layers by weight averaging.
- **FlexiFed** [38] identifies common base layers across client models and clusters personal layers into groups. The same group of personal layers have identical operations and sizes. Then, it fuses the knowledge contained in common base layers and clustered personal layers by weight averaging.
- **InclusiveFL** [29] adopts depth scaling. The shared layers are aggregated via weight averaging. It also distills knowledge from the classifier of the large model to its shallow counterpart by calculating a gradient momentum as the average over updates of the deep layers (pruned in the small model) in the large model and injecting it to the last encoding layer in the small model.
- **DepthFL** [16] scales the large model along the depth and creates local models with multiple classifiers at different depths. The shared layers are averaged for aggregation. It is further equipped with a self-distillation strategy to transfer knowledge among deep and shallow classifiers if available at local. For inference, the client uses the ensemble of all internal classifiers.

Table 1 showcases the downscaling strategies that prior methods are designed for. For architecture-wise scaling, these methods identify common layers (e.g. classification heads) for aggregation.

Federated learning configuration. We evaluate each client model on its respective test data drawn from the client's data distribution. To achieve personalization, for all methods, we fine-tune client models on their local training dataset for one round after receiving parameters from the server. Communication rounds T are set based on the convergence rate of each task. Specifically, we set $T = 50$ rounds for CIFAR-10 and MNLI, $T = 100$ rounds for MNIST, and $T = 500$ rounds for CIFAR-100. At each round, the server randomly selects $|\mathbf{S}_t| = \min(M, 10)$ clients. Since RecipFL trains both small and large models on strong devices for knowledge transfer, we also train both types of models on strong devices for compared methods (i.e., FlexiFed, HeteroFL, InclusiveFL, and DepthFL) to ensure a fair comparison. During evaluation, only the target client model is

[2]https://huggingface.co/bert-base-uncased
[3]https://huggingface.co/distilbert-base-uncased



(a) Impact of data ratio (b) Varying number of weak devices (c) Varying number of strong devices

Figure 5: Exploratory studies. RecipFL exhibits superior scalability and robustness across a range of resource skew scenarios compared to the baselines, consistently enhancing the performance of both strong and weak devices.

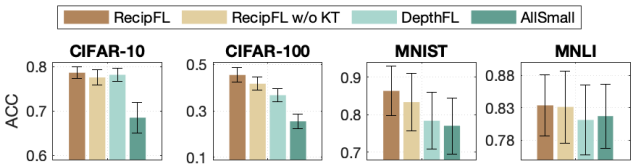


Figure 6: Ablation study: performance of weak devices.

evaluated. The experiments are repeated 5 times. Following [26], we report the average accuracy and standard deviations of the last 20% rounds for strong and weak devices respectively.

5.2 Main Results and Analysis

The experiment results are presented in Table 3. Note that the average accuracy on weak devices may appear higher than that on strong devices since the evaluation is based on every client's data distribution and the weak devices may only have a small subset of classes, making it easier to get higher accuracy. We observe that no scaling strategy consistently outperforms others. For example, width-scaling works better than depth-scaling on CIFAR-10 and CIFAR-100 with the ResNet and DenseNet architectures but it (i.e., the result of HeteroFL) lags depth-scaling on MNIST with LeNet. With architecture-wise scaling on the MNLI dataset, HeteroFL and FlexiFed become equivalent, since the fine-tuning layers, which are positionally aligned across models, have identical sizes (i.e., their scaling ratio is 1). RecipFL outperforms the compared methods across all datasets, regardless of the model scaling strategies, demonstrating its capability to generalize knowledge across different model architectures. Notably, RecipFL shows its ability to improve the model performance on both strong devices and weak devices. Moreover, RecipFL also outperforms the baselines in fine-tuning from the pre-trained weights of BERT and DistilBERT. Details about the performance regarding the communication round are given in Figure 7 in the Appendix. In general, RecipFL achieves a better performance and is more stable than compared methods.

5.3 Ablation Study

In Section 3.3, we introduced the knowledge transfer mechanism within our RecipFL framework to enhance the performance of weak devices. To assess its effectiveness, we craft an ablated version of RecipFL without knowledge transfer, denoted as **RecipFL w/o KT**, and evaluate the model performance on weak devices across four datasets. The CIFAR datasets are tested under the depth scaling setting as examples. We compare the performance of RecipFL, RecipFL w/o KT, AllSmall, and DepthFL (the best-performing baseline with depth scaling). As shown in Figure 6, RecipFL w/o KT already exhibits significant improvements over the naive baseline AllSmall, and it can often outperform the state-of-the-art method DepthFL. However, the comparison between RecipFL and RecipFL w/o KT indicates that integrating knowledge transfer leads to even better small models. The knowledge (i.e., prediction and feature distribution) from strong devices contribute to this improvement.

Table 4: Performance with more diverse device capacities.

Method	Small (LeNet-5)	Medium (ResNet-101)	Large (VGG-16)
AllSmall	69.22±2.16	66.58±1.84	59.06±1.41
ExclusveFL	46.00±3.71	72.38±3.80	79.86±2.52
RecipFL	**70.12±2.33**	**86.37±1.72**	**81.23±0.41**

5.4 More Diverse Device Capacities

RecipFL is not limited to the setup of one large and one small model architecture and can work with diverse device capacities. To support this claim, we conduct an experiment involving a more heterogeneous set of network architectures: 16 small clients training LeNet-5, three medium clients training ResNet-101, and one large client training VGG-16. With RecipFL, strong and medium clients conduct knowledge transfer for models smaller than their respective capacities. We use the CIFAR-10 dataset as an example, with data partitioning of 15% to small clients, 35% to medium clients, and 50% to the large client. The non-IID data on small and medium clients are sampled using Dirichlet distributions $Dir(\alpha = 0.5)$. As shown in Table 4, RecipFL improves performance on each type of device compared to AllSmall and ExclusiveFL.

5.5 Exploratory Studies

To get deeper insights into the performance of the federated systems under various resource skew conditions, we conduct exploratory studies using the MNIST dataset with LeNet models.

Impact of data ratio between two types of devices. We aim to understand how much data on weak devices is necessary to help improve strong devices. We vary the ratio of data allocated to each type of device to the entire dataset among {0.1, 0.5, 0.9}. The number of devices remains the same as in the main experiment, i.e., 500 weak devices and 2 strong devices. Results are presented in Figure 5(a). On both weak and strong devices, models perform better with higher data ratios. It is worth noting that with less than 50% data allocated to strong devices, the performance gap between RecipFL

and ExclusiveFL becomes more evident. This suggests when weak devices hold comparable data amounts to strong devices, they are more likely to contribute significantly to strong devices.

Scalability and skewness. To evaluate the scalability of the federated systems, we first vary the number of weak devices among {250, 500, 750}. The number of strong devices and the data ratio are kept the same as in the main experiments, where the two large devices own 50% of the whole dataset and the weak devices share the rest. The results are shown in Figure 5(b). When increasing the number of weak devices, the strong devices get selected for local updates less frequently. Consequently, within the same communication rounds, the performance of large models degrades. Meanwhile, with fewer data allocated per weak device (as all weak devices collectively share 50% of the dataset), the performance of small models also declines. Despite these challenges, RecipFL demonstrates an impressive ability to memorize model parameters and generalize them across different architectures. As a result, even with reduced client sampling ratios, clients still achieve better performance compared to AllSmall and ExclusiveFL. This suggests RecipFL is more scalable than the baselines. Then, we increase the number of strong devices from 2 to 5 and 50 while keeping weak devices at 500. The strong devices equally share 50% of the entire dataset. As shown in Figure 5(c), RecipFL always achieves better performance than the two baselines on both strong and weak devices. These results highlight the robustness of RecipFL in different levels of skewness.

6 RELATED WORK

Federated learning with heterogeneous models. Traditional federated learning methods [15, 18, 19, 25, 42] have primarily focused on homogeneous models across devices. These methods often fail to address the inherent system heterogeneity found in real-world edge computing environments. Recent studies of federated learning in the context of diverse computational capacities have proposed novel approaches that facilitate collaboration among heterogeneous models [1, 7, 24, 41], focusing on two directions: (1) how to scale the large model and (2) how to effectively aggregate the models with different sizes. In the first direction, methods are proposed to prune the model along depth by pruning the deepest layers [16, 29] or along layer width by scaling the width of hidden channels [6, 23, 31]. In the second direction, typical practices [38] are to identify shared patterns (e.g., layers) in local models and aggregate the common parts. Recent methods like InclusiveFL [29] and DepthFL [16] further leverage knowledge distillation for transferring knowledge among deeper layers and shallow layers to enhance the performance of small models. These approaches have shown promise in accommodating device-specific requirements and resource constraints. However, the reliance on a particular scaling strategy and the naive weight averaging-based aggregation constrain model performance in the presence of resource skew. Our work introduces a more effective way to generalize knowledge across different models by training a graph hypernetwork.

Hypernetworks. A hypernetwork [8] is a neural network that predicts the model parameters of another neural network (i.e., the target network). Hypernetworks have demonstrated the potential in meta-learning scenarios [37], facilitating fast adaptation to new tasks, as they capture the common knowledge among tasks via the weight generation mechanism. Prior work [34] has explored its use in federated learning by training a hypernetwork at the server to generate personalized model weights while preserving the effective parameter-sharing feature of hypernetworks. This previous work uses a linear-structured hypernetwork that only works with homogeneous model architectures. Graph hypernetwork [17, 40] was originally proposed for neural architecture search as it can effectively encode the computational graph information of various neural networks. There has been an initial try on leveraging graph hypernetworks for generating weights across different client models [28]. However, the prior work trains local hypernetworks at clients and aggregates them by weight averaging at the server following a typical federated training process which requires high computational budgets at clients and is impractical for resource-constrained devices. In contrast, RecipFL equips the graph hypernetwork at the server and we design ways to update the graph hypernetwork based on predicted weights and clients' feedback. The computations of hypernetwork are executed only by the server and therefore do not add any additional communication or computation overhead to the edge devices.

7 CONCLUSIONS AND FUTURE WORK

We study the problem of federated learning in the presence of resource skew among devices, specifically, when the majority are weak devices and there are only limited (1 or 2) strong devices. We show that existing methods do not guarantee performance improvement for both types of devices. We propose the RecipFL framework, training a central graph hypernetwork that enables the collaboration of clients with heterogeneous model architectures to fit specific running capacities. RecipFL is agnostic to model scaling strategies and can generalize knowledge about model weights across different neural network architectures. Our experiment results show that RecipFL can outperform state-of-the-art methods with significant margins and demonstrate that even weak devices can contribute effectively to the learning system, providing both devices with an incentive to participate. In future work, we plan to design mechanisms to adaptively adjust the model size in response to the dynamic changes in the running capacity of devices caused by user usage. This will enable efficient utilization of computing resources during learning. Together with our proposed framework, we envision our solutions will create a viable, more powerful, and useable alternative to current large model services, alleviating privacy and efficiency concerns by facilitating edge-based learning without the need to transmit user input to central servers.

ACKNOWLEDGMENTS

This work was supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, NSF CAREER Award 2239440, NSF ProtoOKN Award 2333790, as well as generous gifts from Google, Adobe, and Teradata. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and should not be interpreted as necessarily representing the views, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for government purposes not withstanding any copyright annotation hereon.

REFERENCES

[1] Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. 2022. FedRolex: Model-Heterogeneous Federated Learning with Rolling Sub-Model Extraction. *Advances in Neural Information Processing Systems* 35 (2022), 29677–29690.

[2] Jimmy Ba and Rich Caruana. 2014. Do Deep Nets Really Need to be Deep? *Advances In Neural Information Processing Systems* 27 (2014).

[3] Jonathan Baxter. 2000. A Model of Inductive Bias Learning. *Journal of Artificial Intelligence Research* 12 (2000), 149–198.

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078* (2014).

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).

[6] Enmao Diao, Jie Ding, and Vahid Tarokh. 2020. HeteroFL: Computation and Communication Efficient Federated Learning for Heterogeneous Clients. *arXiv preprint arXiv:2010.01264* (2020).

[7] Xiuwen Fang and Mang Ye. 2022. Robust Federated Learning with Noisy and Heterogeneous Clients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10072–10081.

[8] David Ha, Andrew M. Dai, and Quoc V. Le. 2017. HyperNetworks. In *International Conference on Learning Representations*.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognitio*. 770–778.

[10] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531* (2015).

[11] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. In *International Conference on Machine Learning*. PMLR, 2790–2799.

[12] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. 2019. Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification. *arXiv preprint arXiv:1909.06335* (2019).

[13] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely Connected Convolutional Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4700–4708.

[14] Zhida Jiang, Yang Xu, Hongli Xu, Zhiyuan Wang, Chunming Qiao, and Yangming Zhao. 2022. FedMP: Federated Learning through Adaptive Model Pruning in Heterogeneous Edge Computing. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 767–779.

[15] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. 2020. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In *International conference on machine learning*. PMLR, 5132–5143.

[16] Minjae Kim, Sangyoon Yu, Suhyun Kim, and Soo-Mook Moon. 2023. DepthFL: Depthwise Federated Learning for Heterogeneous Clients. In *The Eleventh International Conference on Learning Representations*.

[17] Boris Knyazev, Michal Drozdzal, Graham W Taylor, and Adriana Romero Soriano. 2021. Parameter Prediction for Unseen Deep Architectures. *Advances in Neural Information Processing Systems* 34 (2021), 29433–29448.

[18] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. 2015. Federated Optimization: Distributed Optimization Beyond the Datacenter. *arXiv preprint arXiv:1511.03575* (2015).

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

[20] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning Multiple Layers of Features from Tiny Images. (2009).

[21] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. 2021. Oort: Efficient Federated Learning via Guided Participant Selection. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 19–35.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[23] Ang Li, Jingwei Sun, Pengcheng Li, Yu Pu, Hai Li, and Yiran Chen. 2021. Hermes: An Efficient Federated Learning Framework for Heterogeneous Mobile Clients. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 420–437.

[24] Ang Li, Jingwei Sun, Xiao Zeng, Mi Zhang, Hai Li, and Yiran Chen. 2021. FedMask: Joint Computation and Communication-Efficient Personalized Federated Learning via Heterogeneous Masking. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. 42–55.

[25] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. *Proceedings of Machine Learning and Systems* 2 (2020), 429–450.

[26] Xin-Chun Li and De-Chuan Zhan. 2021. FedRS: Federated Learning with Restricted Softmax for Label Distribution Non-IID Data. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 995–1005.

[27] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated Graph Sequence Neural Networks. *arXiv preprint arXiv:1511.05493* (2015).

[28] Or Litany, Haggai Maron, David Acuna, Jan Kautz, Gal Chechik, and Sanja Fidler. 2022. Federated Learning with Heterogeneous Architectures using Graph HyperNetworks. *arXiv preprint arXiv:2201.08459* (2022).

[29] Ruixuan Liu, Fangzhao Wu, Chuhan Wu, Yanlin Wang, Lingjuan Lyu, Hong Chen, and Xing Xie. 2022. No One Left Behind: Inclusive Federated Learning over Heterogeneous Devices. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3398–3406.

[30] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Artificial Intelligence and Statistics*. PMLR, 1273–1282.

[31] Chaoyue Niu, Fan Wu, Shaojie Tang, Lifeng Hua, Rongfei Jia, Chengfei Lv, Zhihua Wu, and Guihai Chen. 2020. Billion-Scale Federated Learning on Mobile Clients: A Submodel Design with Tunable Privacy. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–14.

[32] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

[33] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80.

[34] Aviv Shamsian, Aviv Navon, Ethan Fetaya, and Gal Chechik. 2021. Personalized Federated Learning using Hypernetworks. In *International Conference on Machine Learning*. PMLR, 9489–9502.

[35] Chunlin Tian, Li Li, Zhan Shi, Jun Wang, and ChengZhong Xu. 2022. HARMONY: Heterogeneity-Aware Hierarchical Management for Federated Learning System. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 631–645.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *Advances In Neural Information Processing Systems* 30 (2017).

[37] Johannes Von Oswald, Christian Henning, Benjamin F Grewe, and João Sacramento. 2019. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695* (2019).

[38] Kaibin Wang, Qiang He, Feifei Chen, Chunyang Chen, Faliang Huang, Hai Jin, and Yun Yang. 2023. FlexiFed: Personalized Federated Learning for Edge Clients with Heterogeneous Model Architectures. In *Proceedings of the ACM Web Conference 2023*. 2979–2990.

[39] Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. *arXiv preprint arXiv:1704.05426* (2017).

[40] Chris Zhang, Mengye Ren, and Raquel Urtasun. 2018. Graph HyperNetworks for Neural Architecture Search. *arXiv preprint arXiv:1810.05749* (2018).

[41] Jie Zhang, Song Guo, Xiaosong Ma, Haozhao Wang, Wenchao Xu, and Feijie Wu. 2021. Parameterized Knowledge Transfer for Personalized Federated Learning. *Advances in Neural Information Processing Systems* 34 (2021), 10092–10104.

[42] Jiayun Zhang, Xiyuan Zhang, Xinyang Zhang, Dezhi Hong, Rajesh K. Gupta, and Jingbo Shang. 2023. Navigating Alignment for Non-identical Client Class Sets: A Label Name-Anchored Federated Learning Framework. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

A APPENDIX

A.1 More Experiment Results

Performance w.r.t. Communication Rounds. Figure 7 shows the performance of all compared methods with the increase in communication round. We observe that RecipFL often achieve higher accuracy in fewer rounds compared to the baseline methods.

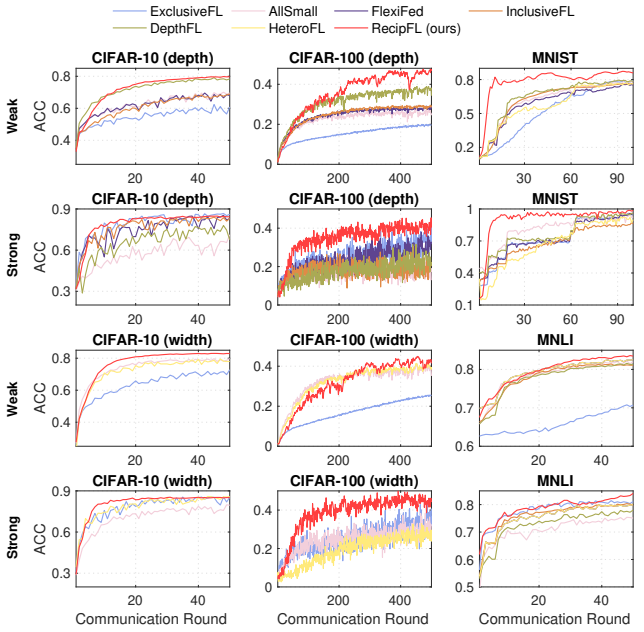


Figure 7: Performance w.r.t. communication round.

A.2 Theoretical Derivations

Notations. We will give the proof of Theorem 4.1 using the results of Baxter [3]. Let us introduce the notations and definitions before we state a key theorem from Baxter (Theorem 18 and Corollary 19) from which the main results of the paper are derived.

Let $\mathcal{X}$ be the input space and $\mathcal{Y}$ be the output space. Let $\mathcal{P}_1, \ldots, \mathcal{P}_M$ be M probability measures on $\mathcal{X} \times \mathcal{Y}$. For every $m = 1, \ldots, M$, sample $(x^{(m)}, y^{(m)})$ from the distribution $\mathcal{P}_m$, and abbreviate $\mathcal{L}_m(\phi) = \ell\left(f_m\left(x^{(m)}; \mathrm{GHN}(\mathcal{G}_m, a_m; \phi)\right), y^{(m)}\right)$, where ϕ represents the parameters of the graph hypernetwork. Define a metric $d_{\mathcal{P}}$ on $\mathbb{R}^d$:

$$d_{\mathcal{P}}(\phi, \phi') = \frac{1}{M}\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{P}}\left|\sum_{m=1}^{M}\mathcal{L}_m(\phi) - \sum_{m=1}^{M}\mathcal{L}_m(\phi')\right|, \tag{12}$$

where $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_M$ is the product probability measure, and $(\boldsymbol{x}, \boldsymbol{y}) = ((x^{(1)}, y^{(1)}), \ldots, (x^{(M)}, y^{(M)}))$. Define the covering number of a subset E of $\mathbb{R}^d$ by closed ball of radius ϵ with respect to the metric $d_{\mathcal{P}}$:

$$\mathcal{N}(\epsilon, E, d_{\mathcal{P}}) = \inf\{n : \exists\phi_1, \cdots, \phi_n, \forall\phi \in E, \exists j,\ d_{\mathcal{P}}(\phi, \phi_j) \le \epsilon\} \tag{13}$$

and the capacity of $E \subset \mathbb{R}^d$ by

$$C(\epsilon, E) = \sup_{\mathcal{P}} \mathcal{N}(\epsilon, E, d_{\mathcal{P}}), \tag{14}$$

where the supremum is taken over all product probability measures on $(\mathcal{X}, \mathcal{Y})^M$. The capacity measures the complexity of the hypothesis space in much the same way as the VC-dimension measures the complexity of a set of Boolean functions. Here our hypothesis space is indexed by $\phi \in \mathbb{R}^d$. Now we are ready to state the theorem from Baxter applied in our RecipFL framework.

Theorem A.1. *Let $\boldsymbol{D} = \{D_m\}_{m=1}^M$ be generated by N independent trials from $(\mathcal{X} \times \mathcal{Y})^M$ according to some product probability measure $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_M$. If*

$$N \ge \max\left\{\frac{4}{M\epsilon^2}\log\frac{4C\left(\frac{\epsilon}{4}, \mathbb{R}^d\right)}{\delta}, \frac{1}{\epsilon^2}\right\}, \tag{15}$$

then

$$\mathbb{P}\left(\boldsymbol{D} : \sup_{\phi}|\mathcal{L}(\phi) - \hat{\mathcal{L}}(\phi, \boldsymbol{D})| > \epsilon\right) \le \delta. \tag{16}$$

Proof of Theorem 4.1. It suffices to bound $C\left(\frac{\epsilon}{4}, \mathbb{R}^d\right)$. Notice that by the Lipschitz assumption on the loss function l, $|\mathcal{L}_m(\phi) - \mathcal{L}_m(\phi')| \le K\|\phi - \phi'\|$ for all $m = 1, \cdots, M$. This implies by (12), for all $\phi, \phi' \in \mathbb{R}^d$,

$$d_{\mathcal{P}}(\phi, \phi') \le \frac{1}{M}\sum_{m=1}^{M}\left|\mathcal{L}_m(\phi) - \mathcal{L}_m(\phi')\right| \le K\|\phi - \phi'\|. \tag{17}$$

So $\|\phi - \phi'\| \le \frac{\epsilon}{K}$ implies $d_{\mathcal{P}}(\phi, \phi') \le \epsilon$. Now take an integer $p > RK\sqrt{d}/\epsilon$ and decompose $[-R, R]^d$ as the union of p^d congruent cubes by dividing $[-R, R]$ into p pieces of equal length. The side length of these cubes is $2R/p$ and so each cube is contained in a ball of radius $R\sqrt{d}/p < \frac{\epsilon}{K}$ centered at the center of the cube. This proves the covering number $\mathcal{N}(\epsilon, E, d_{\mathcal{P}}) \le \lceil RK\sqrt{d}/\epsilon\rceil^d$ for all $\mathcal{P}$. So, $C\left(\frac{\epsilon}{4}, \mathbb{R}^d\right) \le \lceil 4RK\sqrt{d}/\epsilon\rceil^d$. □