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

KEYWORDS
federated learning, edge computing, system heterogeneity

ACM Reference Format:

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)
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 \( \{G_m | m \in M\} \) 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 \( D_m = \{ (x_i^{(m)}, y_i^{(m)}) \}_{i=1}^{N_m} \), where \( x_i^{(m)} \) is the input data and \( y_i^{(m)} \) is the label, and the data distribution of client \( m \) as \( P_m \). Denote \( f \) 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:

\[
\theta^* = \arg\min_{\theta} \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{(x,y) \sim P_m} [f(f_m(x; \theta_m), y)],
\]

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

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
We evaluate client models on their own test data sampled from various (e.g., 5, 20, 50, 500) and 1 or 2 strong devices, allocating 50% updates. The server utilizes the graph hypernetwork to produce personalized client parameters. Algorithm 1 outlines the pseudo code.

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.

Algorithm 1: RecipFL Framework

Input: Communication rounds \( T \), number of selected clients per round \( |S_t| \), local training epochs \( E \), client descriptors \( \{a_m| m \in [M]\} \) and model architectures \( \{G_m| m \in [M]\} \).

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

Server executes:
for \( t = 1, \ldots, T \) do
  Select a subset \( S_t \) of clients at random;
  for \( m \in S_t \) do
    \( \hat{\theta}_m \leftarrow \text{GHN}(G_m, a_m; \phi); \)
    \( \theta_m \leftarrow \text{ClientUpdate}(m, \hat{\theta}_m); \)
    \( \Delta \theta_m \leftarrow \theta_m - \hat{\theta}_m; \)
  Update \( \text{GHN} : \phi \leftarrow \phi - \eta_s \sum_{m \in S_t} (\nabla_{\phi} \theta_m)^T \Delta \theta_m; \)
  return \( \text{GHN}(\cdot; \phi); \)

ClientUpdate\( (m, \hat{\theta}_m)\):
\( \theta_m \leftarrow \hat{\theta}_m; \)
for \( e = 1, \ldots, E \) do
  Partition \( D_m \) into mini-batches \( \{j_i\}_{i=1}^{j_m} (m) \);
  for \( i = 1, \ldots, j_m \) do
    \( \theta_m \leftarrow \theta_m - \eta_c \nabla_{\theta_m} L_m(\theta_m; x_i^{(m)}); \)
  return \( \theta_m \) to server;

model weights \( \{\hat{\theta}_m| m \in S_t\} \), 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 \( D_m \). The training objective at client \( m \) is to minimize the loss:

\[
\arg\min_{\theta} L_m(\theta_m) = \arg\min_{\theta} \frac{1}{N_m} \sum_{i=1}^{N_m} \ell \left( f_m(x_i^{(m)}; \theta_m), y_i^{(m)} \right),
\]

where \( N_m \) is the number of samples in the local dataset \( D_m \). Let \( \eta_c \) be the learning rate for local training at the client. Starting with the initial value \( \theta_m = \hat{\theta}_m \), the client updates \( \theta_m \) as follows:

\[
\theta_m \leftarrow \theta_m - \eta_c \nabla_{\theta_m} L_m(\theta_m).
\]

After local training, the clients send the updated model weights back to the server. The server then calculates the change in local 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, FlexiFed 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 RECIPIFL 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 \( S_t \), to conduct local updates. The server utilizes the graph hypernetwork to produce model weights \( \{\hat{\theta}_m| m \in S_t\} \), 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 \( D_m \). The training objective at client \( m \) is to minimize the loss:

\[
\arg\min_{\theta} L_m(\theta_m) = \arg\min_{\theta} \frac{1}{N_m} \sum_{i=1}^{N_m} \ell \left( f_m(x_i^{(m)}; \theta_m), y_i^{(m)} \right),
\]

where \( N_m \) is the number of samples in the local dataset \( D_m \). Let \( \eta_c \) be the learning rate for local training at the client. Starting with the initial value \( \theta_m = \hat{\theta}_m \), the client updates \( \theta_m \) as follows:

\[
\theta_m \leftarrow \theta_m - \eta_c \nabla_{\theta_m} L_m(\theta_m).
\]

After local training, the clients send the updated model weights back to the server. The server then calculates the change in local
In the encoding phase, a graph neural network (GNN) [27, 33] is employed to conduct \( r \) 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 \( \text{IN}(v) \). The message function is modeled with an MLP shared among all the nodes. The message received by node \( v \) at step \( t \) is:

\[
m^{(t)}_v = \sum_{u \in \text{IN}(v)} \text{MLP}(h^{(t)}_u)
\]

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

\[
h^{(t)}_v = \text{GRU}(h^{(t-1)}_v, m^{(t)}_v)
\]

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 \( \text{MLP}_{\lambda}(\cdot) \) represent the decoder for the \( \lambda \)-type operator. \( \text{MLP}_{\lambda} \) operates on the concatenation of the node embedding and the client embedding, denoted as \( [h^{(t)}_v, a_m] \), and generates parameters for the node. The resulting set of generated weights for the target network is:

\[
w = \{w_v | v \in \mathcal{V}\} = \{\text{MLP}_{\lambda}(h^{(t)}_v, a_m) | v \in \mathcal{V}\}
\]

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 models at the strong device \( m \) as \( f^s_m \) and \( f^l_m \) respectively and the corresponding model parameters as \( \theta^s_m \) and \( \theta^l_m \). After training the large model \( f^l_m \), 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 $p_{m}^S$ and $p_{m}^L$ respectively. The optimization objective for training small model $f^m_S$ on strong device $m$ is to minimize the following loss:

$$L^S_m(\theta) = \frac{1}{n} \sum_{i=1}^{n} [CE(f^m_S(x_i; \theta^S_m), y_i) + CE(f^m_S(x_i; \theta^S_m), f^m_L(x_i; \theta^L_m)) + D_{KL}(p^L_i \| p^S_i)].$$

(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 $D_m = \{(x^{(m)}_i, y^{(m)}_i)\}_{i=1}^{N}$ for some natural number $N \geq 1$, i.e. we sample uniformly $N$ training data from each data distribution $P_m$ on client $m$ for $m = 1, \cdots, M$.

Assume the loss function $\ell$ takes value in $[0, 1]$, or equivalently with rescaling, $t$ is bounded. Let $d$ be the dimension of the hypernetwork parameter $\phi$ and assume $\phi \in [-R, R]^d$ for some large $R > 0$. Finally, assume the loss $t$ is Lipschitz with respect to $\phi$ with Lipschitz constant $K > 0$, i.e. $|t(f_m(x; GHN(G_m, a_m; \phi)), y) - t(f_m(x; GHN(G_m, a_m; \phi')), y)| \leq K ||\phi - \phi'||$ for all $x, y$ and $m = 1, \cdots, M$. Here $\|\cdot\|$ denotes the Euclidean distance on $\mathbb{R}^d$. Define the expected loss as:

$$\mathcal{L}(\phi) = \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{(x, y) \sim P_m}[\ell(f_m(x; (G_m, a_m; \phi)), y)].$$

(10)

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

$$N \geq \max \left\{ \frac{4d}{M} \log \left[ \frac{4RK \sqrt{d}}{\varepsilon} \right] + \frac{4}{M} \log \left[ \frac{4}{\delta} \frac{1}{\varepsilon^2} \right], \right\},$$

(11)

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

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>
<thead>
<tr>
<th>Scaling Strategy</th>
<th>HeteroFL</th>
<th>InclusiveFL</th>
<th>FlexiFed</th>
<th>DepthFL</th>
<th>ReciPFL (ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>depth-wise</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>width-wise</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>architecture-wise</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Scaling</th>
<th>Method</th>
<th>CIFAR-10 (Strong)</th>
<th>CIFAR-10 (Weak)</th>
<th>CIFAR-100 (Strong)</th>
<th>CIFAR-100 (Weak)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth</td>
<td>AllSmall</td>
<td>64.34±2.14</td>
<td>68.50±3.42</td>
<td>17.86±2.56</td>
<td>23.56±3.11</td>
</tr>
<tr>
<td></td>
<td>ExclusiveFL</td>
<td>84.85±1.85</td>
<td>59.11±4.22</td>
<td>32.21±3.81</td>
<td>19.22±2.07</td>
</tr>
<tr>
<td></td>
<td>FlexiFed [38]</td>
<td>82.86±1.77</td>
<td>67.66±3.93</td>
<td>28.60±3.48</td>
<td>27.84±2.98</td>
</tr>
<tr>
<td></td>
<td>InclusiveFL [29]</td>
<td>83.22±0.47</td>
<td>67.66±3.14</td>
<td>18.98±3.49</td>
<td>28.71±2.87</td>
</tr>
<tr>
<td></td>
<td>DepthFL [16]</td>
<td>73.90±1.49</td>
<td>78.16±1.48</td>
<td>22.08±3.58</td>
<td>36.83±2.87</td>
</tr>
<tr>
<td></td>
<td>RecipFL</td>
<td>85.28±0.22</td>
<td>78.65±1.35</td>
<td>41.63±2.24</td>
<td>45.52±3.12</td>
</tr>
<tr>
<td>Width</td>
<td>AllSmall</td>
<td>82.86±1.77</td>
<td>78.90±2.87</td>
<td>29.86±3.32</td>
<td>37.90±2.83</td>
</tr>
<tr>
<td></td>
<td>ExclusiveFL</td>
<td>85.96±1.97</td>
<td>70.65±3.99</td>
<td>32.22±6.66</td>
<td>24.49±3.52</td>
</tr>
<tr>
<td></td>
<td>HeteroFL [6]</td>
<td>84.76±1.19</td>
<td>77.93±2.92</td>
<td>26.51±2.70</td>
<td>39.05±2.82</td>
</tr>
<tr>
<td></td>
<td>RecipFL</td>
<td>85.06±0.13</td>
<td>82.88±1.29</td>
<td>43.64±2.84</td>
<td>42.00±3.88</td>
</tr>
</tbody>
</table>

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.

```
<table>
<thead>
<tr>
<th>Dataset</th>
<th># of devices</th>
<th>Data allocation</th>
<th>Large model</th>
<th># of parameters</th>
<th>Pretrained?</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>1</td>
<td>5</td>
<td>50%</td>
<td>10%</td>
<td>ResNet-18</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>1</td>
<td>50</td>
<td>50%</td>
<td>1%</td>
<td>DenseNet-121</td>
</tr>
<tr>
<td>MNIST</td>
<td>2</td>
<td>500</td>
<td>25%</td>
<td>0.1%</td>
<td>LeNet-5</td>
</tr>
<tr>
<td>MNLI</td>
<td>1</td>
<td>20</td>
<td>50%</td>
<td>2.5%</td>
<td>BERT</td>
</tr>
</tbody>
</table>

RecipFL   | 11M          | 450K            | 444K        | ✗              |
| ExclusiveFL | 1M          | 258K            | 276K        | ✗              |
| InclusivF | 44K          | 5.6K (scaled in depth & width) | ✗ |
| lexiFed   | 110M         | 67M (DistilBERT) | ✔            |
```

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:

- **DepthFL** [16] adopts depth 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 \(S_t = \min(M, 10)\) 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
how few davids improve one goliath: federated learning in resource-skewed edge computing environments

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.

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(α = 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.

Table 4: Performance with more diverse device capacities.

<table>
<thead>
<tr>
<th>Method</th>
<th>Small (LeNet-5)</th>
<th>Medium (ResNet-101)</th>
<th>Large (VGG-16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllSmall</td>
<td>69.22±2.16</td>
<td>66.58±1.84</td>
<td>59.06±1.41</td>
</tr>
<tr>
<td>ExclusiveFL</td>
<td>46.00±3.71</td>
<td>72.38±3.80</td>
<td>79.86±2.52</td>
</tr>
<tr>
<td>RecipFL</td>
<td>70.12±2.33</td>
<td>86.37±1.72</td>
<td>81.23±0.41</td>
</tr>
</tbody>
</table>

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, 23, 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 Proto-OKN Award 233790, 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


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

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

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

$$N(\varepsilon, E, d_\mathcal{P}) = \inf \{ n : \exists \phi_1, \ldots, \phi_n, \forall \phi \in E, \exists j, d_\mathcal{P}(\phi, \phi_j) \leq \varepsilon \}$$  

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

$$C(\varepsilon, E) = \sup_{\mathcal{P}} N(\varepsilon, E, d_\mathcal{P}),$$  

where the supremum is taken over all product probability measures on $(X, \mathcal{Y})^M$. 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 $D = \{D_m\}_{m=1}^M$ be generated by $N$ independent trials from $(X \times \mathcal{Y})^M$ according to some product probability measure $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_M$. If

$$N \geq \max \left\{ \frac{4 \epsilon^2}{M \mu^2}, \frac{4C(\frac{\varepsilon}{4}, \mathbb{R}^d)}{\delta}, \frac{1}{e^2} \right\},$$

then

$$\mathbb{P} \left( D : \sup_\phi |L(\phi) - \hat{L}(\phi, D)| > \varepsilon \right) \leq \delta.$$  

**Proof of Theorem 4.1.** It suffices to bound $C \left( \frac{\varepsilon}{4}, \mathbb{R}^d \right)$. Notice that by the Lipschitz assumption on the loss function $\ell$, $|L_m(\phi) - L_m(\phi')| \leq K \|\phi - \phi'\|$ for all $m = 1, \ldots, M$. This implies by (12), for all $\phi, \phi' \in \mathbb{R}^d$,

$$d_\mathcal{P}(\phi, \phi') \leq \frac{1}{M} \sum_{m=1}^{M} |L_m(\phi) - L_m(\phi')| \leq K \|\phi - \phi'\|.$$  

So $\|\phi - \phi'\| \leq \frac{\varepsilon}{4}$ implies $d_\mathcal{P}(\phi, \phi') \leq \varepsilon$. Now take an integer $p > R K \sqrt{d}/\varepsilon$ 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 $R \sqrt{d} / p < \sqrt{d}$ centered at the center of the cube. This proves the covering number $N(\varepsilon, E, d_\mathcal{P}) \leq [RK \sqrt{d} / \varepsilon]^d$ for all $\mathcal{P}$. So, $C \left( \frac{\varepsilon}{4}, \mathbb{R}^d \right) \leq [4RK \sqrt{d} / \varepsilon]^d$. □