EFFICIENT SPLIT-MIX FEDERATED LEARNING FOR ON-DEMAND AND IN-SITU CUSTOMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

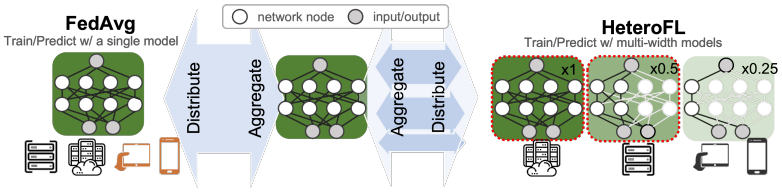
Federated learning (FL) provides a distributed learning framework for multiple participants to collaborate learning without sharing raw data. In many practical FL scenarios, participants have heterogeneous resources due to disparities in hardware and inference dynamics that require quickly loading models of different sizes and levels of robustness. The heterogeneity and dynamics together impose significant challenges to existing FL approaches and thus greatly limit FL's applicability. In this paper, we propose a novel Split-Mix FL strategy for heterogeneous participants that, once training is done, provides *in-situ customization* of model sizes and robustness. Specifically, we achieve customization by learning a set of base sub-networks of different sizes and robustness levels, which are later aggregated on-demand according to inference requirements. This split-mix strategy achieves customization with high efficiency in communication, storage, and inference. Extensive experiments demonstrate that our method provides better in-situ customization than the existing heterogenous-architecture FL methods.

1 INTRODUCTION

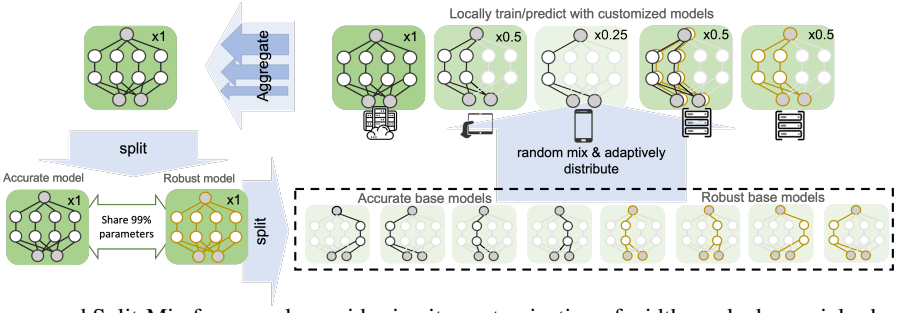
Federated learning (Konečný et al., 2015) is a distributed learning paradigm that leverages data from remote participants and aggregates their knowledge without requiring their raw data to be transferred to a central server, thereby largely reducing the concerns from data security and privacy. FedAvg (McMahan et al., 2017) is among the most popular federated instantiations, which aggregates knowledge by averaging homogeneous model parameters.

In many practical federated learning scenarios, the participants usually exhibit a variety of *heterogeneity*. The training datasets from participants can be non-*i.i.d.* and therefore lead to data heterogeneity, which has received extensive attention in the community (Li et al., 2020b; Fallah et al., 2020; Hong et al., 2021; Li et al., 2021). In addition, the participants can also have resource heterogeneity, i.e., participants have different hardware resources available, such as memory, computing power, and network bandwidth (Ignatov et al., 2018). For example, in a learning task for face recognition, clients may use different types of devices (e.g., desktop computers, different generations of tablets or smartphones) to participate in learning, which thereby leads to drastic differences in computational resources for training models, and in sizes of models that can be loaded to the memory and transferred through the network. The disparity of hardware resources results in varying budgets in both training and inference stages (Xu et al., 2021). To accommodate different model sizes, one can turn to more flexible architectures trained by distillation from ensemble (Lin et al., 2020), partial model averaging (Diao et al., 2021), or directly combining predictions (Shi et al., 2021).

Yet besides heterogeneity, the deployment of models usually has a great amount of *dynamics*. One type of dynamics is directly related to the resources of participants' devices. For example, memory consumptions and computational capabilities may vary drastically between training and inference stages, and the same dynamics in hardware can also happen during inference (Yu et al., 2018; Xu et al., 2021), depending on how the resource allocation of the running programs is prioritized on a participant's device. Another example of dynamics is the requirements of properties of the model, such as adversarial robustness. In autonomous driving, for instance, cloudy and rainy weather can lower the quality of camera and other sensors, and therefore one would like to quickly switch to a more robust prediction model, as compared to the one used on sunny days. The dynamics along with the low latency requirements for the inference, altogether demand a model that can be customized *in situ*,



(a) Illustration of FedAvg (McMahan et al., 2017) and a heterogenous-architecture variant (HeteroFL) (Diao et al., 2021). The single model trained by FedAvg cannot fit into budget-insufficient devices. Though the HeteroFL trains a slimmable model, the wider models can be under-trained due to limited data in budget-sufficient devices.



(b) The proposed Split-Mix framework provides in-situ customization of widths and adversarial robustness to address heterogeneity and dynamics, enabling efficient training and inference. In this example, we use a subnet with 1/4 channels (or widths) per layer as a base model for model-width customization. For simplicity, we denote it $\times 0.25$ net, and a $\times 1$ net can be split into 4 $\times 0.25$ base models.

Figure 1: Comparison of traditional and proposed methods.

meaning that we can instantly tailor our predictive model to a proper size that fits our available budget, and promptly adjust its robustness. Most existing federate learning models used a rather static model architecture that prevents such customization. A naive solution is to simultaneously train multiple models with wanted architectures and properties, which however may lead to prohibitive training cost and latency of switching models in a budget-limited device. Moreover, the trade-off between model adversarial robustness and generalization performance can hardly be reconcilable (Tsipras et al., 2019; Merchant et al., 2020), which further complicates the customization.

To address the aforementioned challenges from heterogeneity and dynamics, we study a novel *Split-Mix* federated learning approach for *in-situ customization*, which follows a communication-efficient paradigm and yields a model whose size and robustness can be efficiently customized, without additional training. Specifically, we first **split** the complete knowledge in a large model into several small base sub-networks (shards) according to model widths and robustness levels. To complete the knowledge, we let the base models be fully trained on all clients. To provide customized models, we **mix** selected base models to construct desired widths and required robustness. We illustrate the main idea in Fig. 1b. Overall, our contributions can be summarized in three folds:

- Within the domain of heterogeneous federated learning, we are the first to study training a model with the capability of *in-situ customization* with heterogeneous local computation budgets. Illustrated in Fig. 1a, we show that conventional static-model solutions, such as FedAvg, cannot provide customized models for dynamic client environments, and that its heterogeneous-architecture variant (HeteroFL) suffers from under-training in its large models due to local budget constraints.

- To address the challenge, we propose a novel Split-Mix framework that aggregates knowledge from heterogeneous clients into a width- and robustness-adjustable model structure. Remarkably, due to fewer parameters and modular nature, our framework is not only efficient in federated communication, but also efficient in storage, model loading and inference.

- Empirically, we demonstrate that the performance of the proposed method is better than other FL baselines under heterogeneous budget constraints. Moreover, we show its effectiveness when facing the challenge of data heterogeneity, i.e., clients' training datasets suffer from distributional shifts.

2 Related Work

Heterogeneous Federated Learning. Federated learning with budget-insufficient devices has attracted a great amount of attention. For example, *FedDistill* (Lin et al., 2020) used logit averages to distill a network of the same size or several prototypes, which will be communicated with users. FedDistill made an assumption that the central server has access to a public dataset of the same learning task, which is impractical for many applications. He et al. (2020) introduced a distillation-based method after aggregating private representations from all participants. The method closely resembles centralized learning because all encoded samples are gathered, and however it is less efficient when local clients have large data dimensions or sample sizes. Importantly, the method may not transfer adversarial robustness knowledge through the intermediate representations due to the decoupling of the input and prediction. On the other hand, *HeteroFL* (Diao et al., 2021) avoids distillation, allowing participants to train different prototypes and sharing parameters among prototypes to reduce redundant parameters. However, HeteroFL also reduces the samples available for training each prototype, which leads to degraded performance. *FedEnsemble* (Shi et al., 2021) is technically related to the proposed approach, which uses ensemble of diverse models to accommodate non-*i.i.d.* heterogeneity. The authors showed that combining multiple base models trained simultaneously in FL can outperform a single base model in testing. A critical difference between the proposed approach and FedEnsemble comes from the challenging problem setting of constrained heterogeneous computation budgets. Moreover, we show that with fewer model parameters and training time, the ensemble of a subset of trained base models could outperform a single model of the same width but trained in a heterogeneous-architecture way.

Customizable Models. To our best knowledge, this is the first paper discussing in-situ customization in federated learning and here we review similar concepts in central learning. First, customization of robustness and accuracy was discussed by Wang et al. (2020), where an adjustable-conditional layer together with decoupled batch-normalization were used. The conditional layer enables continuous trade-off but also brings in over 50% more parameters. In comparison, a simple weighted combination without additional parameters is used in our method and is very efficient in both communication and inference. In terms of model complexity, a series of work on dynamic neural networks were proposed to provide instant, adaptive and efficient trade-off between accuracy and latency (mainly related to model complexity) of neural networks at the inference stage. Typically, sub-path networks of different complexity (Liu & Deng, 2018) or sub-depth networks (Huang et al., 2018; Wu et al., 2019; Wang et al., 2018; Zhang et al., 2021) are trained together. However, due to the large memory footprint brought by a constant number of channels per layer, the memory footprint at inference is barely reduced. To address the challenge, slimmable neural network (SNN) (Yu et al., 2018) was proposed to train networks with adaptive layer widths. Distinct from SNN, we consider a more challenging scenario with distributed and non-sharable data and heterogeneous client capabilities.

3 Problem Setting

The goal of this work is to develop a federated learning (FL) strategy, that yields a global model, which can be instantly customized for dynamic needs. Formally, FL minimizes the objective $\frac{1}{\sum_k |D_k|}\sum_{k=1}^{K}\sum_{(x,y)\in D_k} L(f(x;W),y)$ with distributed data $\{D_k\}_{k=1}^{K}$ across K participants/clients, where L is loss function, e.g., cross-entropy loss, f is a model parameterized by W, and x, y are a sample and a label, respectively. In this paper, we consider two kinds of customization: *adjustable model sizes* to trade off between execution complexity and accuracy, *adjustable robustness* for dynamic security sensitivity. Though other types of customization could be considered, for example, neural-architecture search or fairness, we believe the two types considered are quite representative for both architecture and non-architecture knowledge (Wang et al., 2020).

When training customizable/adjustable models, significant challengs arise from heterogeneity among clients. In this paper, we consider the following: **1) Heterogeneous computational budgets during training** of clients constrain the maximal complexity of local models. The complexity of deep neural networks can be defined in multiple dimensions, like depths or widths (Zagoruyko & Komodakis, 2017; Tan & Le, 2019). In this paper, we consider the width of hidden channels, because it can significantly reduce not only the number of model parameters but also the layer output cache. Thus, we assume clients have confined width capabilities $\{R_k \in (0,1]\}_{k=1}^{K}$, defined by width ratios w.r.t. the original model, i.e., $\times R_k$ net as presented in Fig. 1b. Similar to FedAvg and HeteroFL, the

same architecture is used by clients and therefore the model width can be tailored according to local budgets. **2) Imbalance computation budgets.** In many applications, there are usually a significant number of devices with insufficient computational budgets (Enge, 2021). Without loss of generality, we assume exponentially distributed budgets in uniformly divided client groups: $R_k = (1/2)^{\lceil 4k/K \rceil}$, i.e., the first group with $1/4$ clients is capable for a 1-width and the rest are for $0.5, 0.25, 0.125$-widths respectively. Such a budget distribution could result in additional challenges in the presence of data heterogeneity, e.g., non-*i.i.d.* features, $D_k \sim \mathcal{D}_i$ for ordered domain i, since training a single model on multiple domains (Li et al., 2020b) or models in a single domain (due to budget constraints) (Hong et al., 2021) are known to be suboptimal on average performance. Finally, following the standard FL setting (McMahan et al., 2017), only model parameters can be shared to protect privacy.

In summary, the problem setting has challenges from 1) resource heterogeneity, where disparity in computation budgets will significantly affect the training of large models because they can only be trained scarce budget-sufficient clients; 2) data heterogeneity, where the under-trained large models may perform poorly on unseen in-domain or out-of-domain samples.

4 METHOD

To provide efficient *in-situ* customization, we introduce a simple yet powerful principle, Split-Mix: *shatter complete knowledge into smaller pieces, and customize by flexible formations of pieces.* Based on the principle, we propose customizations of model size and robustness in the following.

4.1 CUSTOMIZE MODEL SIZE

For motivation, we first set up a non-*i.i.d.* FL experiment to compare model performance with/without budget constraint, where the same 6-domain DomainNet images are uniformly distributed into 50 clients for both cases (refer to experimental details in Section 5).

Case 1: FL without budget constraint. First, we individually train networks of different widths by FedAvg. We see that the slimmer networks converge slower and are less generalizable, even though they are trained on all clients (solid lines in Fig. 2). The results justified the motivation of training wider networks than slimmer ones.

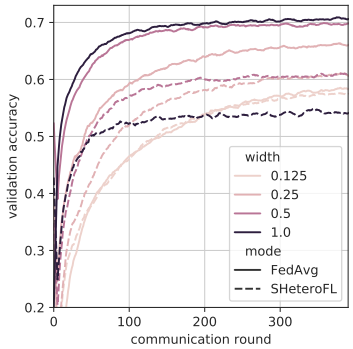


Figure 2: Convergence of different-width models on DomainNet. For experiment details, see Section 5.1.

Case 2: FL with budget constraint. Following the above budget constraint, i.e., $R_k = (1/2)^{\lceil 4k/K \rceil}$, we deploy HeteroFL to train budget-compatible prototype models locally. We note that data are not fully used by HeteroFL: each model prototype is only trained by $1/4$ of clients, when clients of the $R_k = 1$ budget can actually afford all slimmer models. Therefore, we extend HeteroFL to a *slimmable* version (SHeteroFL) by training all affordable prototypes locally. As shown in Fig. 2, wider models (e.g., $\times 1$ net) converge to validation accuracy lower than not only the FedAvg counterparts, but also the slimmer $\times 0.25$ net, showing that the widest model may not be a good candidate model.

Intuitively, as a consequence of the budget constraint, fewer training clients are able to participate in the learning of wider networks, where the reduced sample size or model size greatly impacted the generalization. Since FL does not allow global data sharing to mitigate the issue, the key is to study how the scarcity of data or smaller width affects model generalization in FL. We hypothesize that a model may pick up signals from more features for reasoning a class if trained with more data. For validation, we investigate the features that draw significant attention of different networks in Fig. 3. We found that both FedAvg and SHeteroFL $\times 1$ nets extract richer features than the slimmer $\times 0.125$ net. This may partially explain why the slimmer network generalizes poorly. On the other side, though the SHeteroFL $\times 1$ net has the same capacity as the FedAvg $\times 1$ net and thereby extracted similarly-rich features, its accuracy is lower than FedAvg. This suggests that the features learned from limited data could be less generalizable. For example, the SHeteroFL loses attention of edges of the feather in the fifth image. Thus, more diverse or higher-quality features are desired. The importance of features for generalization was also echoed by a recent study (**?**), The study showed that encouraging neural networks to extract diverse features can boost testing accuracy.

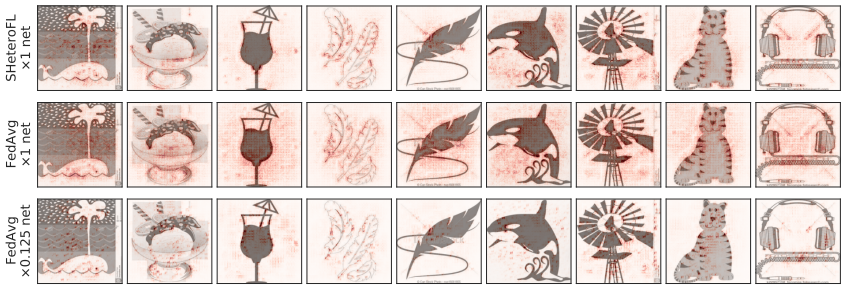


Figure 3: Gradient saliency ($\frac{\partial L}{\partial x}$) (Sundararajan et al., 2017) for AlexNet predictions on DomainNet test data. Trained by different widths/methods, models reason their predictions by different features.

Motivated by the above observations, we propose to train multiple base models from different random initializations to encourage their diversity. The diverse bases likely extract different features from the same image (see Fig. 4). The overall FL algorithm is summarized in Algorithm 3. To craft a wide model, we combine the output of a number of base models until the size of the ensemble reaches the same as the number of channels. Therefore, the ensemble predicts based on a variety of features, resembling an integrated wide network. To facilitate these base models to extract generalizable features, we want to train base models in all clients. Inspired by FedEnsemble (Shi et al., 2021), we randomly sample a set of bases for each client in Algorithm 1. In each client, we speed up the local training of multiple independent base models by direct parallelization (see Algorithm 2). Note however that the parallelism is limited by the maximal width in local, as wider channels result in larger intermediate layer cache (activations). We thus limit the total channels of ensemble by client's budgets. Note that since the federated training processes of base models are independent without interference, the training could be as stable as FedAvg with partial participants. Different from sequential training, the parallelism speeds up the local training and reduces the possible client blocking time on communication. In addition, slimmer base models can conform a variety of client budgets while maintaining the same model width.

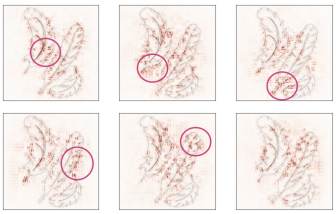


Figure 4: Different gradient saliency generated by 6 $\times 0.125$ base networks on the same image. More images in Fig. 12.

Algorithm 1 SampleBaseModels(P, p, n, W)

1: **if** $p > |P|$ **then** Shuffle P and $p \leftarrow 1$
2: Initialize $W = \{w_{P[p],r}\}$
3: **if** n>1 **then**
4: Uniformly sample $n-1$ values into S from $P \backslash \{P[p]\}$ without replacement
5: $W \leftarrow W \cup \{w_{i,r}, \forall i \in S\}$
6: Return $W, p+1$

Algorithm 2 LocalTrain(W_k, D_k, E, η)

1: Initialize models $\hat{W}_k$ by W_k
2: **for** $e \in 1, \cdots, E$ **do**
3: **for** batch data (x, y) in D_k **do**
4: **for** $\hat{w}_{i,r} \in \hat{W}_k$ in parallel **do**
5: $\hat{w}_{i,r} \leftarrow \hat{w}_{i,r} - \eta \frac{\partial L(f(x;\hat{w}_{i,r}),y)}{\partial \hat{w}_{i,r}}$
6: Return $\hat{W}_k$

4.2 Customize Adversarial Robustness

In many existing robustness learning approaches, model robustness and generalization are typically irreconcilable, and a trade-off between the two has to be determined when applying robust models. In this section, we study the customization of model robustness, and specifically *adversarial robustness*. The adversarial robustness of a model reflects its prediction accuracy when subtle and prediction-flipping adversarial noise presents. Recently, *adversarial training* (AT) (Madry et al., 2018) becomes a popular approach that uses on-the-fly adversarial samples as augmentation to improve adversarial robustness. Formally, AT minimizes the following augmented loss:

$$L(f) = (1-\lambda_n) L_{CE}(f(x), y) + \lambda_n \max_{\|\delta\|_\infty \leq \epsilon} L_{CE}(f(x+\delta), y), \tag{1}$$

where δ is a subtle adversarial noise which maximizes the loss in a given constraint ϵ. The addition of δ transfers a clean sample x into an *adversarial sample*. In Eq. (1), the first term characterizes the cross-entropy loss (L_{CE}) on clean samples while the later one on adversarial samples. The

Algorithm 3 Federated Split-Mix Learning

Input: Client datasets $\{D_k\}_{k=1}^K$, the number of total communication rounds T, $M = \lfloor 1/r \rfloor$ randomly initialized base $\times r$ nets parameterized by $\{w_{i,r}\}_{i=1}^M$, client budgets $\{R_k\}_{k=1}^K$, the number of local epochs E, learning rates $\{\eta_t\}_{t=1}^T$

Initialize $\{w_{i,r}^0\}_{i=1}^M$, model indexes $P = \text{Shuffle}([1, \cdots, M])$ and current index $p = 1$
for round $t \in \{1, \cdots, T\}$ **do**
  Initialize $W^t \leftarrow \{w_{i,r}^t \leftarrow 0\}_{i=1}^M$ and aggregation weights $c_i = 0$ for $i \in \{1, \ldots, M\}$
  for $k \in \{1, \cdots, K\}$ **do**
    Sample base models $W_k^{t-1}, p \leftarrow \text{SampleBaseModels}(P, p, \lfloor R_k/r \rfloor, W^{t-1})$
    Send W_k^{t-1} to client k and train $\hat{W}_k^t \leftarrow \text{LocalTrain}(W_k^{t-1}, D_k, E, \eta_t)$
    Aggregate $\hat{w}_{i,r}^{t,k} \in \hat{W}_k^t$ to the server: $w_{i,r}^t \leftarrow w_{i,r}^t + \hat{w}_{i,r}^{t,k}|D_k|$, $c_i \leftarrow c_i + |D_k|$
  Server update $W^t \leftarrow \{w_{i,r}^t \leftarrow w_{i,r}^t / c_i\}_{i=1}^M$
Sort $W^T = \{w_{i,r}^T\}_{i=1}^M$ by the descending order of the validation accuracy of $w_{i,r}^T$
Output the customizable model $W^T = \{w_{i,r}^T\}_{i=1}^M$
Customize an R-width model by $F(x; W^T) = \frac{1}{K_R} \sum_{i=1}^{K_R} f(x; w_{i,r}^T)$ where $K_R = \lfloor R/r \rfloor$

hyper-parameter λ_n trades off robustness and accuracy. When $\lambda_n = 0$ or 1, the optimization yields a *standard-training* (ST) model or an AT model, respectively. Since an adversarially-trained model is commonly less accurate in predicting standard images (Tsipras et al., 2019), there is usually no such a sweet point of λ_n simultaneously maximizing robustness and accuracy, and one typically needs to carefully gauge the trade-off according to the demand of robustness in specific application context.

Splitting and sharing parameters. Since standard performance and adversarial robustness are irreconcilable, we can directly use two separated ST and AT models to maximally capture the each property. But do we really need two totally separated models? Intuitively, the two models share some common knowledge, given that an adversarial image share a large part of common features with its original version. As introduced by (Xie & Yuille, 2019; Xie et al., 2020), sharing all parameters except the batch-normalization (BN) layers can maximize robustness and accuracy by expertised BNs, respectively. Accordingly, we propose to split BN layers (instead of the whole model) into two sub-components: one for standard performance and the other for robustness. At training time, the first cross-entropy loss of Eq. (1) is computed with clean BN (BN_c) merely and the second adversarial loss is computed with noised BN (BN_n). The FL local training is elaborated in Algorithm 4. As the two BNs are decoupled, there is no more trade-off in loss Eq. (1) and thereby we choose $\lambda_n = 0.5$ to balance their effects on parameter updates.

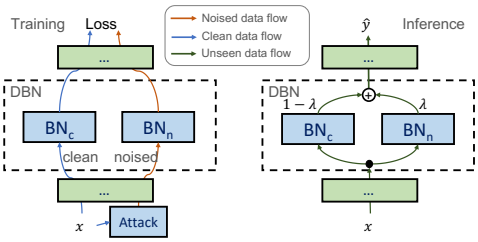


Figure 5: Illustration of dual batch-normalization (DBN) in training and inference. The BN_c and BN_n are for clean and noised samples.

Layer-wise mixing. After training, the problem is how to mix the two models (with different BNs) for prediction. A straightforward solution is averaging their outputs. However, forwarding memory footprint will be doubled in this way, as the two models have to be both executed separately. To avoid the cache of double size of intermediate outputs, a gate function can be used to adaptively switch between two BNs like (Liu et al., 2020). Inspired by the method, we further propose a simple parameter-free method by weighted-averaging outputs of each BN layer (see Fig. 5):

$$\text{DBN}(x) = (1 - \lambda)\text{BN}_c(x) + \lambda \text{BN}_n(x), \tag{2}$$

given a BN-layer input x. We note that the averaging strategy is entirely training-free and does not use extra parameters, and the customization weight λ is intuitive for trade-offs between SA and RA.

Lastly, it is remarkable that the DBN structure is rather lightweight in terms of model complexity. As investigated in (Yu et al., 2018) (Table 2), the parameters of BN is no more than 1% in popular deep architectures, e.g., ResNet or MobileNet. Therefore, we can plug DBN into base models in place of BN, replace Algorithm 2 with Algorithm 4, and jointly customize robustness and model widths.

5 EMPIRICAL STUDIES

We design experiments to compare the proposed method against FL classification benchmarks. **Configurations.** For *i.i.d* configuration, we use CIFAR10 dataset (Krizhevsky, 2009) with preactivated ResNet (PreResNet18) (He et al., 2016). CIFAR10 contains over $50,000$ 32×32 images of 10 classes. The CIFAR10 data are uniformly split into 100 clients. For non-*i.i.d* configuration, we use Digits with a CNN defined and DomainNet datasets (Li et al., 2020b) with AlexNet extended with BN layers after each convolutional or linear layer (Li et al., 2020b). **Data.** The first dataset is a subset (30%) of Digits, a benchmark for domain adaption (Peng et al., 2019b). Digits has 28×28 images and serves as a commonly used benchmark for FL (Caldas et al., 2019; McMahan et al., 2017; Li et al., 2020a). The dataset includes 5 different domains: MNIST (Lecun et al., 1998), SVHN (Netzer et al., 2011), USPS (Hull, 1994), SynthDigits (Ganin & Lempitsky, 2015), and MNIST-M (Ganin & Lempitsky, 2015). The second dataset is DomainNet (Peng et al., 2019a) processed by (Li et al., 2020b), which contains 6 distinct domains of large-size 256×256 real-world images: Clipart, Infograph, Painting, Quickdraw, Real, Sketch. Each domain of Digits (or DomainNet) are split into 10 (or 5) clients, and therefore 50 (or 30) clients in total. We defer other details such as hyper-parameters to Appendix A, and focus on discussing the results.

5.1 CUSTOMIZE MODEL SIZES

In this section, we evaluate the proposed Split-Mix on tasks of customizing model sizes through adjusting model widths. Recall that in Section 3, we assume a specific heterogeneous training budget to facilitate our discussion, such that one client can only train models within a maximal width, and the resource distribution is imbalance among clients: $R_k = (1/2)^{\lceil 4k/K \rceil}$. Following the common FL setting, we do not consider algorithms that use public data or representation sharing in our baselines.

Baselines. As an ideal upper bound but a memory-incompatible baseline, we (re-)train networks from scratch by FedAvg to obtain individual models with different widths. The state-of-the-art heterogeneous-architecture FL method is *HeteroFL* (Diao et al., 2021), which trains different slim models in different clients. For a fair comparison, we extend HeteroFL with bounded-slimmable training in clients who can afford the larger models, named *SHeteroFL*. For example, if a client in HeteroFL can afford $\times 0.5$ net, then the client meanwhile trains $\times 0.25$ net and other smaller subnets in slimmable training manner (Yu et al., 2018) by SHeteroFL.

Convergence. In Fig. 6, we compare the convergence curves of full-width models on three datasets. For FedAvg, the budget-compatible width is $\times 0.125$. Both for SHeteroFL and Split-Mix, the full-width is $\times 1$. Compared to baselines, the proposed Split-Mix converges faster, largely due to the splitting strategy. Specifically, because all base models are independently trained, the convergence of each base model mainly depends on how frequently they are trained. When enough clients participate FL, the training frequency of the $\times 1$ net is more than one, as all bases will be selected at least once in a communication round.

Performance. In Table 1, we compare the test accuracy of different model widths. We measure the latency in terms of MACs (number of multiplication-and-addition operations) and model size in parameter numbers. With the same width, our method outperforms SHeteroFL, using much fewer parameters and thus conducts inference in much lower latency. Remarkably, compared to the best model by SHeteroFL, e.g., 81.8% $\times 1$ net in CIFAR10, Split-Mix uses only 1.8% parameters and 1.6% MACs ($\times 0.125$ net) to reach a similar level of test accuracy. We notice that SHeteroFL has a much lower test accuracy with $\times 0.125$ net on the CIFAR10 dataset. By investigating the loss curves, we find that the inference between parameter-shared different prototypes results in unstable convergence of the $\times 0.125$ net. Attributed to the independent splitting, our method is more stable

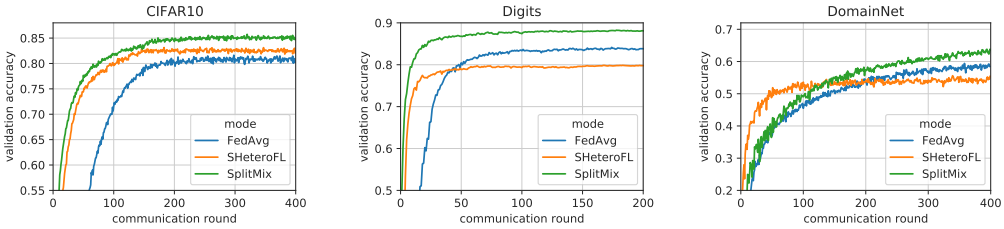


Figure 6: Validation accuracy of budget-compatible full-wdith nets by iterations.

Table 1: Test results of customizing model width. MACs and the number of parameters are counted at inference time. Grey texts indicate that the training cannot conform the predefined budget constraint. The 'M' after metric values means $\times 10^6$.

	Individual FedAvg			SHeteroFL			Split-Mix (ours)		
width	Acc	MACs	#Params	Acc	MACs	#Params	Acc	MACs	#Params
				CIFAR10 *i.i.d* FL					
$\times 0.125$	80.4%	9.0M	0.2M	71.3%	9.0M	0.2M	**81.3%**	9.0M	0.2M
$\times 0.25$	~~83.0%~~	~~35.3M~~	~~0.7M~~	79.6%	35.3M	0.7M	**83.5%**	**18.0M**	**0.4M**
$\times 0.5$	~~88.9%~~	~~139.9M~~	~~2.8M~~	81.8%	139.9M	2.8M	**85.0%**	**36.0M**	**0.7M**
$\times 1$	~~90.9%~~	~~556.5M~~	~~11.2M~~	81.2%	556.5M	11.2M	**85.9%**	**72.1M**	**1.4M**
				Digits non-*i.i.d* FL					
$\times 0.125$	**85.2%**	1.2M	0.2M	84.4%	1.2M	0.2M	84.2%	1.2M	0.2M
$\times 0.25$	~~87.4%~~	~~3.7M~~	~~0.9M~~	**86.7%**	3.7M	0.9M	86.1%	**2.4M**	**0.4M**
$\times 0.5$	~~88.8%~~	~~13.0M~~	~~3.6M~~	86.1%	13.0M	3.6M	**88.1%**	**6.1M**	**1.3M**
$\times 1$	~~89.3%~~	~~48.0M~~	~~14.2M~~	81.6%	48.0M	14.2M	**89.1%**	**19.1M**	**4.9M**
				DomainNet non-*i.i.d* FL					
$\times 0.125$	61.6%	24.8M	0.9M	59.4%	24.8M	0.9M	**63.5%**	24.8M	0.9M
$\times 0.25$	~~69.3%~~	~~75.5M~~	~~3.6M~~	62.3%	75.5M	3.6M	**67.3%**	**49.5M**	**1.8M**
$\times 0.5$	~~72.1%~~	~~254.8M~~	~~14.3M~~	62.2%	254.8M	14.3M	**68.7%**	**99.0M**	**3.6M**
$\times 1$	~~72.5%~~	~~925.0M~~	~~57.1M~~	56.2%	925.0M	57.1M	**69.8%**	**198.1M**	**7.2M**

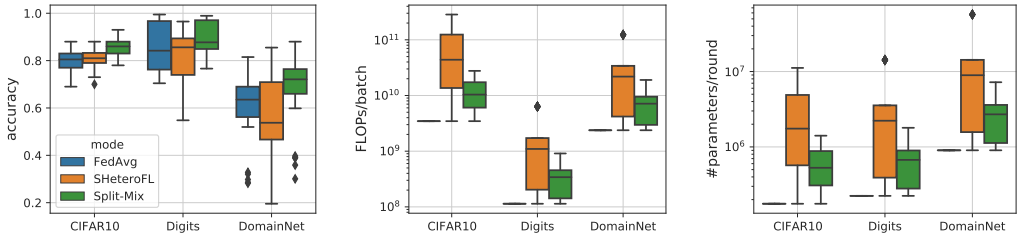


Figure 7: Client-wise statistics of test accuracy, training and communication efficiency by budget constraints. The MACs quantify the complexity of one batch optimization in a client, and the number of parameters per round round are the ones uploaded to (or downloaded from) a server. Test accuracy is by the full-width networks. The results of FedAvg are from budget-compatible $\times 0.125$ nets.

on convergence with different widths. Remarkably, Split-Mix only requires 12.7% parameters and 19.8% MACs (by $\times 1$ ensemble) to achieve the comparable accuracy as the $\times 1$ individual network on Digits. Results on CIFAR10 and DomainNet show a potential limitation of our method. The accuracy of Split-Mix is not comparable with the wider individual models due to the locally limited complexity. However, the limitation is more from the problem setting itself, and will not undermine our advantage in budget-limited FL, since wide individual models cannot be trained in this case.

In addition to comparisons of inference on average, we also demonstrate the statistics of test accuracy and the training and communication efficiency in Fig. 7. Conforming the budget constraints, our method outputs more accurate full-width models, transfer fewer parameters per round and execute fewer multiplication-and-add operations for gradient descent than SHeteroFL, either in terms of average or variance. Because only the individual $\times 0.125$ net can fit into the budget constraint, FedAvg requires the least MACs and parameters, which however significantly sacrifices the final accuracy.

To understand why Split-Mix outperforms SHeteroFL, we investigate the total percentage of parameters that can be trained in each domain, in Fig. 8 (Left). We count the total parameters that were ever trained in clients of a domain during the learning. Thanks to the base-model sampling strategy (i.e., Algorithm 1), Split-Mix allows all base models, rather than a subset, to be trained on all clients. On the other hand, varying client budgets greatly impacted SHeteroFL by limiting the width of models trained in budget-insufficient clients, e.g., in the clipart, infograph and painting domains. Hence, SHeteroFL leaves a large amount of parameters under-trained and suffers from larger accuracy losses in the three domains and wider models. In comparison, Split-Mix not only has less accuracy drop but is also more stable in all domains.

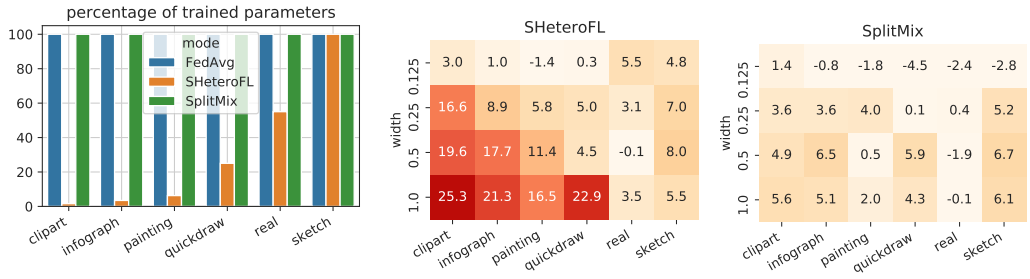
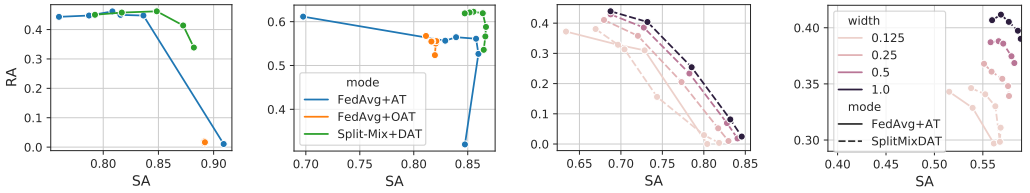


Figure 8: Per domain in DomainNet, the total percentage of parameters that are locally trained (the left figure) and the accuracy (%) drops compared to FedAvg individual models (right two figures).



(a) CIFAR10 (b) Digits (c) CIFAR10 (d) DomainNet

Figure 9: Trade-off between robust accuracy (RA) and standard accuracy (SA) with full width (a,b) and customizable widths (c,d).

5.2 Customize robustness

Training and evaluation. For local AT, we use an n-step projected gradient descent (PGD) attack (Madry et al., 2018) with a constant noise magnitude ϵ. Following (Madry et al., 2018), we set $(\epsilon, n) = (8/255, 7)$, and attack inner-loop step size $2/255$, for training, validation, and test. For simplicity, we temporarily *relax the budget constraint* R_k *and let the base model be* $\times 1$ *net.* For comparison, we extend FedAvg with AT which yields individual models optimized with different trade-off variables, i.e., $\lambda_n \in \{0, 0.1, 0.2, 0.3, 0.5, 1\}$. Also, we extend FedAvg with state-of-the-art in-situ trade-off method, OAT (Wang et al., 2020), as a baseline. Split-Mix+DAT is an extension of Split-Mix by the proposed DBN-based AT in Algorithm 4. We evaluate and contrast models in two metrics: *standard accuracy (SA)* on the clean test samples and *robust accuracy (RA)* on adversarial images generated from the clean test set by the PGD attack. Both metric values are averaged by users. We evaluate the trade-off effectiveness by comparing the RA-SA curves, which is better approaching the right-upper corner. To plot the curves for FedAvg+OAT and Split-Mix+DAT, we vary their condition variable λ in $\{0, 0.2, 0.5, 0.8, 1\}$.

Trade-off curves are presented in Fig. 9 (a) and (b). Since the naive extension of OAT with FedAvg adopts heterogeneous objectives and over-parameterization of conditional layers which suffers from convergence issues, its adversarial training does not converge and RA is incredibly poor in some cases. As a result, the trade-off curve is not smooth. Instead, the proposed Split-Mix+DAT method has a smoother trade-off curve without heavy conditional training or over-parameterization. By training in one pass, Split-Mix+DAT even outperforms and is more efficient than the FedAvg+AT baselines.

Joint customization of width and robustness under budget constraints. Now we consider the width customization and the training budgets as Section 5.1. Due to the constraint, FedAvg can only train $\times 0.125$ net. We omit OAT for its unstable convergence and use SplitMixDAT as a short name of Split-Mix+DAT. In Fig. 9 (c) and (d), the trade-off curves with different widths are depicted. As the width increases, both RA and SA of SplitMixDAT are improved, when they are smoothly traded off. The results demonstrate the flexibility and the modular nature of our method.

6 Conclusion

In this paper, we proposed a novel federated learning approach for in-situ and on-demand customization to address challenges arise from resource heterogeneity and inference dyanmics. We proposed a Split-Mix strategy that efficiently transfer clients' knowledge to collaboratively learn a customizable model. Extensive experiments demonstrate the effectiveness of the principle in adjusting model widths and robustness when much fewer parameters are used compared to baselines.

REFERENCES

Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv:1812.01097 [cs, stat]*, December 2019.

Enmao Diao, Jie Ding, and Vahid Tarokh. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. In *International Conference on Learning Representations*, 2021.

Eric Enge. Mobile vs. desktop usage in 2020. https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage, March 2021.

Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning: A meta-learning approach. In *Advances in Neural Information Processing Systems*, June 2020.

Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning*, pp. 1180–1189. PMLR, June 2015.

Chaoyang He, Murali Annavaram, and Salman Avestimehr. Group knowledge transfer: Federated learning of large cnns at the edge. *Advances in Neural Information Processing Systems*, November 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Computer Vision and Pattern Recognition*, 2016.

Junyuan Hong, Zhuangdi Zhu, Shuyang Yu, Zhangyang Wang, Hiroko H. Dodge, and Jiayu Zhou. Federated adversarial debiasing for fair and transferable representations. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pp. 617–627, New York, NY, USA, August 2021. Association for Computing Machinery.

Gao Huang, Danlu Chen, Tianhong Li, and Felix Wu. Multi-scale dense networks for resource efficient image classification. *International Conference on Learning Representations*, pp. 14, 2018.

Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, May 1994.

Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pp. 0–0, 2018.

Jakub Konečný, Brendan McMahan, and Daniel Ramage. Federated optimization:distributed optimization beyond the datacenter. *arXiv:1511.03575 [cs, math]*, November 2015.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10713–10722, 2021.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *Conference on Systems and Machine Learning Foundation (MLSys)*, April 2020a.

Xiaoxiao Li, Meirui Jiang, Xiaofei Zhang, Michael Kamp, and Qi Dou. Fedbn: Federated learning on non-iid features via local batch normalization. In *International Conference on Learning Representations*, September 2020b.

Tao Lin, Lingjing Kong, Sebastian U. Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. In *Advances in Neural Information Processing Systems*, June 2020.

Aishan Liu, Shiyu Tang, Xianglong Liu, Xinyun Chen, Lei Huang, Zhuozhuo Tu, Dawn Song, and Dacheng Tao. Towards defending multiple adversarial perturbations via gated batch normalization. *arXiv:2012.01654 [cs]*, December 2020.

Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. *arXiv:1701.00299 [cs, stat]*, March 2018.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *International Conference on Learning Representations*, 2018.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282, April 2017.

Amil Merchant, Barret Zoph, and Ekin Dogus Cubuk. Does data augmentation benefit from split batchnorms. *arXiv:2010.07810 [cs]*, October 2020.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1406–1415, 2019a.

Xingchao Peng, Zijun Huang, Yizhe Zhu, and Kate Saenko. Federated adversarial domain adaptation. In *International Conference on Learning Representations*, September 2019b.

Naichen Shi, Fan Lai, Raed Al Kontar, and Mosharaf Chowdhury. Fed-ensemble: Improving generalization through model ensembling in federated learning. *arXiv:2107.10663 [cs, stat]*, July 2021.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*, pp. 3319–3328. PMLR, July 2017.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 6105–6114. PMLR, May 2019.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *International Conference on Learning Representations*, September 2019.

Haotao Wang, Tianlong Chen, Shupeng Gui, Ting-Kuei Hu, Ji Liu, and Zhangyang Wang. Once-for-all adversarial training: In-situ tradeoff between robustness and accuracy for free. *Advances in Neural Information Processing Systems*, November 2020.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. *ECCV*, July 2018.

Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. *arXiv:1711.08393 [cs]*, January 2019.

Cihang Xie and Alan Yuille. Intriguing properties of adversarial training at scale. *International Conference on Learning Representations*, December 2019.

Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan Yuille, and Quoc V. Le. Adversarial examples improve image recognition. *CVPR*, April 2020.

Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. A first look at deep learning apps on smartphones. *arXiv:1812.05448 [cs]*, January 2021.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv:1812.08928 [cs]*, December 2018.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv:1605.07146 [cs]*, June 2017.

Linfeng Zhang, Chenglong Bao, and Kaisheng Ma. Self-distillation: Towards efficient and compact neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

A EXPERIMENTS

In this section, we provide more details about our experiments and additional evaluation results.

A.1 EXPERIMENTAL CONFIGURATIONS

In Algorithm 4, we elaborate the local training of DBN. Specifically, each BN is trained independent with different input samples. The maximization problem in Algorithm 4 can be solved by an n-step projected gradient descent, and is commonly known as *PGD attack* (Madry et al., 2018).

Algorithm 4 LocalTrain(W_k, D_k, E, η) with DBN and adversarial training

Initialize models $\hat{W}_k$ by W_k
for $e \in \{1, \cdots, E\}$ **do**
  for mini-batch $B = \{(x, y)\}$ in D_k **do**
    for $\hat{w}_{i,r} \in \hat{W}_k$ in parallel **do**
      Set f to use clean BN
      $L \leftarrow \frac{1}{|B|} \sum_{(x,y)\in B} L_{CE}(f(x; \hat{w}_{i,r}), y)$
      Set f to use noise BN
      $\tilde{B} = \emptyset$
      for $x \in B$ **do**
        Perturb $\tilde{x} = x + \delta$ with $\delta \leftarrow \arg\max_{\|\delta\|_\infty \leq \epsilon} L_{CE}(f(x + \delta; \hat{w}_{i,r})), y)$
        $\tilde{B} \leftarrow \tilde{B} \cup \{(\tilde{x}, y)\}$
      $L \leftarrow \frac{1}{2} \left\{ L + \frac{1}{|\tilde{B}|} \sum_{(\tilde{x},y)\in\tilde{B}} [L_{CE}(f(\tilde{x}; \hat{w}_{i,r}), y)] \right\}$
      $\hat{w}_{i,r} \leftarrow \hat{w}_{i,r} - \eta \frac{\partial L}{\partial \hat{w}_{i,r}}$
Return $\hat{W}_k$

Data. Both CIFAR10 and Digits are 10-way classification tasks. We follow the non-*i.i.d* benchmark of Li et al. (2020b) to extract 10 classes from DomainNet, which is publicly available in FedBN codes. To illustrate the multiple-domain datasets, we sample several images from Digits and DomainNet datasets in Fig. 10.

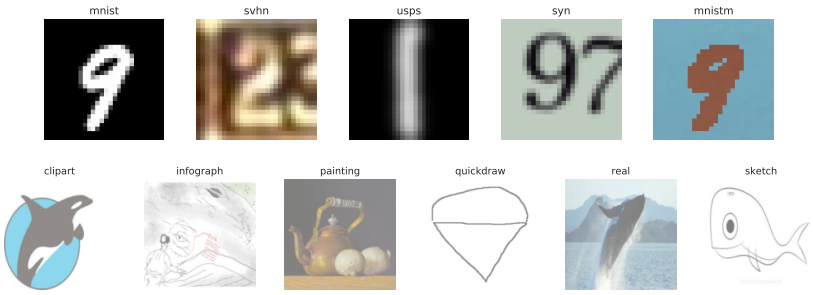


Figure 10: Sample images from multiple domain datasets.

Hyper-parameters. In general, for local optimization we use stochastic gradient descent (SGD) with 0.9 momentum and 5×10^{-4} weight decay. Dataset specific settings are stated as follows. CIFAR10: Following HeteroFL (Diao et al., 2021), we train with 5 local epochs and 400 global communication rounds. Globally, we initialize the learning rate as 0.01 and adjust the learning rate at 150, 250 communication rounds with a scale rate of 0.1. Locally, we use a larger batch size of 128, to speed up the training in simulation. Digits: We use a cosine annealing learning rate decaying from 0.1 to 0 across 400 global communication rounds. SGD is executed with one epoch for each local client. DomainNet: We use a constant learning rate 0.01 and run 400 communication rounds in total. Similar to Digits, SGD is executed with one epoch for each local client.

Table 2: Network architecture for Digits dataset.

Layer	Details
feature extractor	
conv1	Conv2D(64, kernel size=5, stride=1, padding=2)
bn1	DBN2D, RELU, MaxPool2D(kernel size=2, stride=2)
conv2	Conv2D(64, kernel size=5, stride=1, padding=2)
bn2	DBN2D, ReLU, MaxPool2D(kernel size=2, stride=2)
conv3	Conv2D(128, kernel size=5, stride=1, padding=2)
bn3	DBN2D, ReLU
classifier	
fc1	FC(2048)
bn4	DBN2D, ReLU
fc2	FC(512)
bn5	DBN1D, ReLU
fc3	FC(10)

Table 3: Network architecture for DomainNet dataset.

Layer	Details
feature extractor	
conv1	Conv2D(64, kernel size=11, stride=4, padding=2)
bn1	DBN2D, ReLU, MaxPool2d(kernel size=3, stride=2)
conv2	Conv2D(192, kernel size=5, stride=1, padding=2)
bn2	DBN2D, ReLU, MaxPool2d(kernel size=3, stride=2)
conv3	Conv2D(384, kernel size=3, stride=1, padding=1)
bn3	DBN2D, ReLU
conv4	Conv2D(256, kernel size=3, stride=1, padding=1)
bn4	DBN2D, ReLU
conv5	Conv2D(256, kernel size=3, stride=1, padding=1)
bn5	DBN2D, ReLU, MaxPool2d(kernel size=3, stride=2)
avgpool	AdaptiveAvgPool2d(6, 6)
classifier	
fc1	FC(4096)
bn6	DBN1D, ReLU
fc2	FC(4096)
bn7	DBN1D, ReLU
fc3	FC(10)

Network architectures. Architectures of modified AlexNet (for DomaiNet) and CNN (for Digits) can be found in (Li et al., 2020b) and public codes [1]. For reader's reference, we provide the layer details in in Tables 2 and 3. For the convolutional layer (Conv2D or Conv1D), the first argument is the number channel. For a fully connected layer (FC), we list the number of hidden units as the first argument. The implementation of the preactivated ResNet can be found in the repository of HeteroFL (Diao et al., 2021).

A.2 Customize model sizes

Batch-normalization. As pointed out by Diao et al. (2021), HeteroFL relies on mini-batch batch-normalization to stabilize the training and compute the statistics afterwards. To eliminate the impact of biased mini-batch BN statistics, we let all compared algorithms using the same strategy. Another advantage of such strategy is on federation of multi-domain clients. Li et al. (2020b) showed that using local batch-normalization helps model personalization for local features. As min-batch BN statistics simulate the local BN idea, which should enjoy the similar benefit.

[1] https://github.com/med-air/FedBN

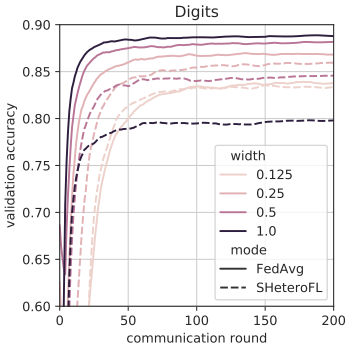


Figure 11: Convergence of different-width models. Details of the experiment are enclosed in Section 5.1.

Figure 12: Gradient saliency of $\times 0.125$ nets trained from different random initializations. The different parameter initializations make slim networks to pay attention diverse features.

Saliency is extracted using the `captum` package[2]. We provide more visualization of the saliency of networks of varying widths.

[2] https://github.com/pytorch/captum/tree/master

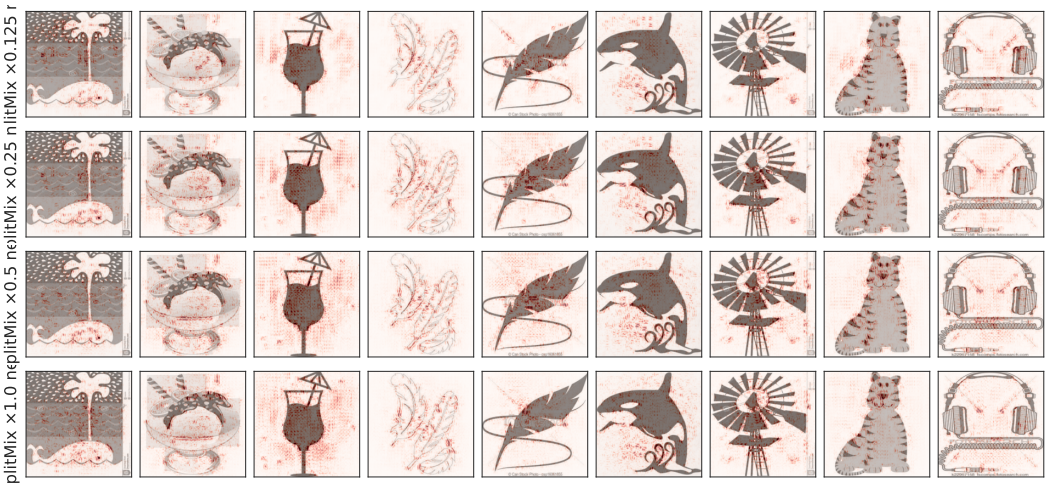


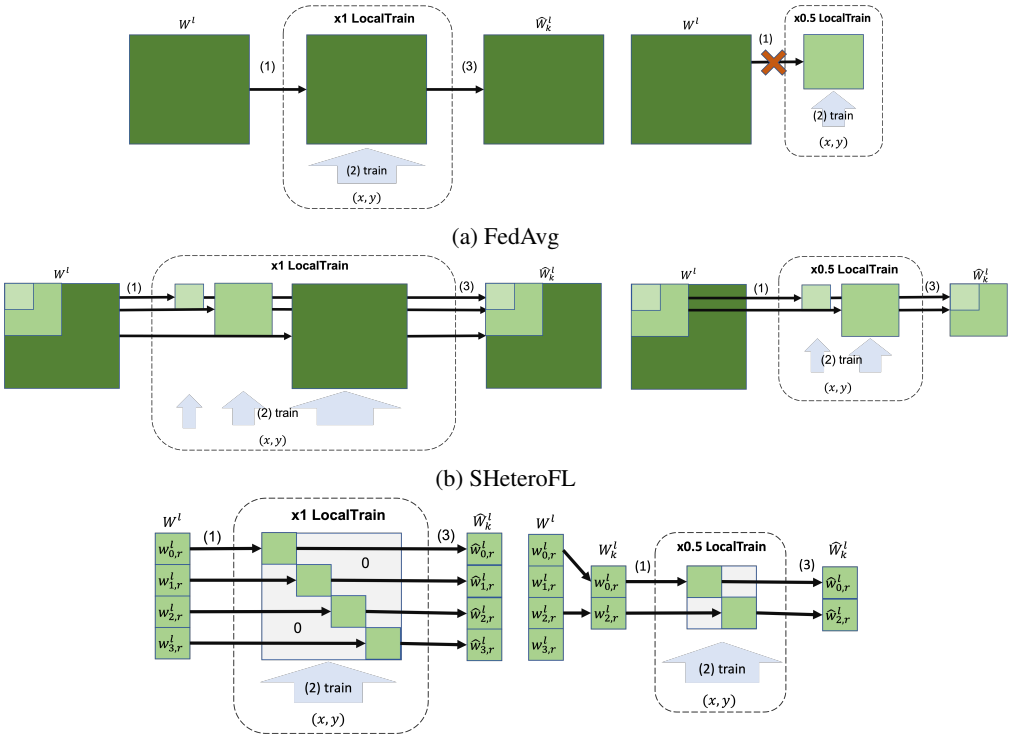
Figure 13: Gradient saliency of different ensemble widths. Wider networks are presented in the lower rows. Wider networks learn more and informative features. For example, in the second column, The widest model reduces some random features but pay more attention to the object edges.

B ADDITIONAL EXPERIMENTS FOR REBUTTAL

Content in this section is added in the period of rebuttal, which will be re-organized after acceptance.

B.1 PARALLEL IMPLEMENTATION OF SPLIT-MIX AND CONVERGENCE

In this section, we elaborate on the implementation and efficiency of Split-Mix. In Fig. 14, we conceptually compare the training by three methods, when two clients capable of training $\times 1$ and $\times 0.5$ are considered. FedAvg can train the $\times 1$ net on the $\times 1$-capable client but not on the $\times 0.5$-capable client. Through parameter sharing among different model widths, SHeteroFL can train multiple widths in a sequential manner and can fit into $\times 0.5$-capable clients. Unlike SHeteroFL, Split-Mix trains four base models in one parallel pass and therefore is the most efficient and flexible method. Remarkably, in the worst case, training $\times 1$ net of Split-Mix is as efficient as FedAvg and more efficient than SHeteroFL, if all the base weights, $w^l_{0,r}, \dots, w^l_{3,r} \in \mathbb{R}^{r \times r}$ (where $r = 0.25$ here), are embedded into a $4r \times 4r$ weight matrix[3]. When base models are embedded into a full net, non-trainable parameters can be masked out (grey areas) to avoid interference between base models.



(a) FedAvg

(b) SHeteroFL

(c) Split-Mix

Figure 14: Illustration of training weight matrices on a $\times 1$-net-capable or $\times 0.5$-net-capable client. (1) Download the global weight matrix W^l of layer l or a selected subset W^l_k. (2) Train weights on a batch data (x, y). (3) Upload trained weight matrix $\hat{W}^l_k$.

With the aforementioned implementation, we compare the convergence versus the wall-clock time in Fig. 15. We implement all algorithms in `PyTorch 1.4.1` run on a single NVIDIA RTX A5000 GPU and a 104-thread CPU. Fig. 15 shows the elapsed computation time from the initialization to a maximal number of iterations. The maximal number of iterations is set to be the same for all methods, such that it is easier to compare the stopping time. In Fig. 15, Split-Mix is much more efficient than the SheteroFL. Note that FedAvg can only train the $\times 0.125$ net which includes much fewer parameters, For this reason, Split-Mix is slightly slower than FedAvg, but the degradation of efficiency trades in better accuracy than FedAvg.

[3]For the simplicity of notations, we assume the width of layer l is r, though the width could vary by layer in general.

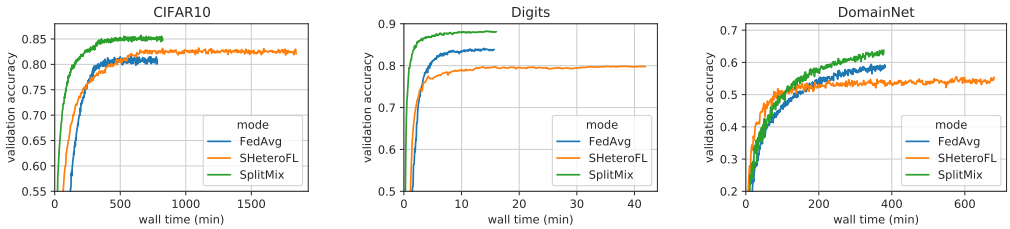


Figure 15: Validation accuracy of the budget-compatibly-widest nets by wall-clock time. All algorithms are run with the same number of iterations (400, 200 and 400, respectively).

B.2 Experiments with class non-i.i.d FL

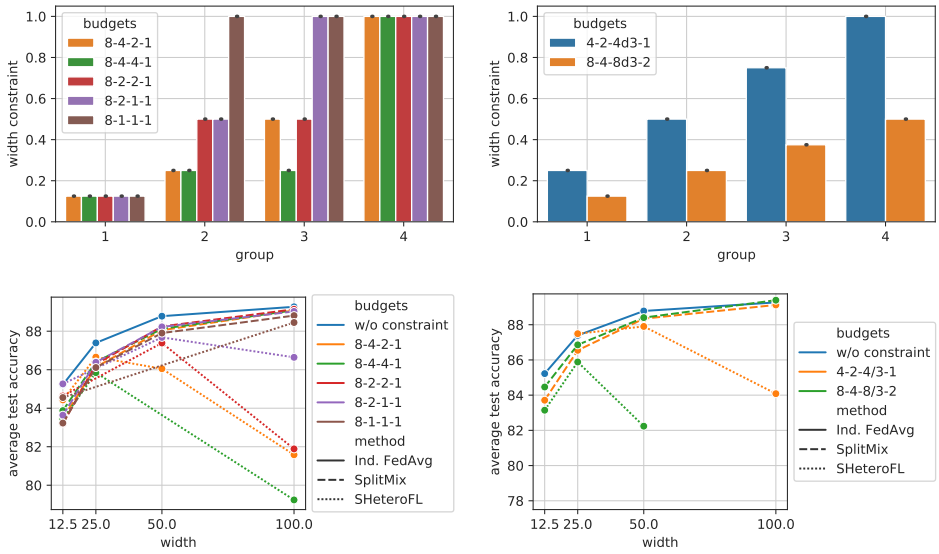
In addition to feature non-*i.i.d* FL settings, we experiment with class non-*i.i.d* FL where each client will randomly select 3 out of 10 classes from the CIFAR10 dataset. To mitigate the effect of missing classes locally, we use the same masked cross-entropy as introduced by HeteroFL (Diao et al., 2021). Results are included in Table 4. We observe a great drop in the accuracy compared to the *i.i.d* experiments, which is a common phenomenon that non-iid FL will perform worse globally. The similar performance degradation was observed in (Diao et al., 2021), as well. Besides the degradation, our method outperforms baselines consistently. Furthermore, our method performs better in larger widths, whose performance approaches that of unconstrained individual FedAvg.

Table 4: Test results of customizing model width on the class non-*i.i.d* CIFAR10 dataset.

	Individual FedAvg			SHeteroFL			Split-Mix (ours)		
width	Acc	FLOPs	#Params	Acc	FLOPs	#Params	Acc	FLOPs	#Params
				CIFAR10 class non-*i.i.d* FL					
$\times 0.125$	**53.4%**	9.0M	0.1M	42.0%	9.0M	0.1M	50.9%	9.0M	0.1M
$\times 0.25$	~~54.4%~~	~~35.3M~~	~~0.4M~~	47.7%	35.3M	0.4M	**52.7%**	**18.0M**	**0.2M**
$\times 0.5$	~~54.5%~~	~~139.9M~~	~~1.4M~~	48.5%	139.9M	1.4M	**54.3%**	**36.0M**	**0.4M**
$\times 1$	~~54.6%~~	~~556.5M~~	~~5.6M~~	45.8%	556.5M	5.6M	**55.0%**	**72.1M**	**0.7M**

B.3 More budget distributions

In our experiments, we generally use an exponential budget distribution: $R_k = (1/2)^{\lceil 4k/K \rceil}$. Though the distribution represents the imbalance between the budget-sufficient and budget-insufficient clients, real-world applications may encounter a wider variety of budget distributions. Thus, we extend our problem assumption to budget distributions with *more budget-sufficient clients* where we let more groups to have $\times 1$ or $\times 0.5$ net training capability and with *step-increase budgets* where we increase budgets by a fixed step (e.g., $\times 0.25$). Fig. 16 reports the per-width accuracy for Split-Mix and SHeteorFL on the Digits dataset. For SHeteroFL, we only report the evaluated performance on trained withs. In other words, if the maximal width is $\times 0.5$, we will not report results of the $\times 1$ net. For Individually-trained FedAvg (Ind. FedAvg), models individually trained for each width are reported, which ignores the width constraints by users and therefore only serves as reference upper bounds. Regardless of the budget distributions, for instance, more budget-sufficient clients (Fig. 16a) or non-exponential distributions (Fig. 16b), Split-Mix outperforms SHeteroFL with larger widths.



(a) More budget-sufficient clients

(b) Step-increase budget distributions

Figure 16: Vary the budget distribution. The training budgets, i.e., width constraints, are depicted in the upper figures by group. The budget distribution name, for example, 8-4-2-1, means $\times 1/8$, $\times 1/4$, $\times 1/2$ and $\times 1$ width constraints for each group, respectively. The lower figures compare the performance of trained models with customized withs.