Resilient and Communication Efficient Learning for Heterogeneous Federated Systems

Zhuangdi Zhu [1] **Junyuan Hong** [1] **Steve Drew** [2] **Jiayu Zhou** [1]

Abstract

The rise of Federated Learning (FL) is bringing machine learning to edge computing by utilizing data scattered across edge devices. However, the heterogeneity of edge network topologies and the uncertainty of wireless transmission are two major obstructions of FL's wide application in edge computing, leading to prohibitive convergence time and high communication cost. In this work, we propose an FL scheme to address both challenges simultaneously. Specifically, we enable edge devices to learn self-distilled neural networks that are readily prunable to arbitrary sizes, which capture the knowledge of the learning domain in a nested and progressive manner. Not only does our approach tackle system heterogeneity by serving edge devices with varying model architectures, but it also alleviates the issue of connection uncertainty by allowing transmitting part of the model parameters under faulty network connections, without wasting the contributing knowledge of the transmitted parameters. Extensive empirical studies show that under system heterogeneity and network instability, our approach demonstrates significant resilience and higher communication efficiency compared to the state-of-the-art.

1. Introduction

Federated Learning (FL) is a decentralized machine learning scheme that eliminates private data sharing on participating devices. Recent years witnessed effervescent development of FL in varied domains, including healthcare (Rieke et al., 2020), computer vision (Liu et al., 2020), natural language processing (Hard et al., 2018; McMahan et al., 2017), and Internet of things (IoT) (Khan et al., 2021; Du et al., 2020), to name just a few. The rapid adoption and deployment of edge computing have enabled computing to be even closer to the source of the data and users. The growing demand for privacy-aware and low-latency machine learning at the edge makes FL a natural fit.

The diversities among participating devices and their network topologies are phenomenal, which imposed significant challenges of *statistical* and *system* heterogeneity to FL. The statistical heterogeneity in FL has been extensively tackled by techniques such as regularized optimization (Dinh et al., 2020), customized model aggregation (Wang et al., 2020), and domain-invariant representation leanring (Zhu et al., 2021; Hong et al., 2021c). In comparison, system heterogeneity, which is induced by significant gaps in memory capacities and transmission bandwidth among edge devices, is under-explored. Moreover, traditional FL hinges on reliable connections, where model parameters are transmitted between edge devices and a central server without packet loss. This prerequisite can be prohibitive for practical edge-based applications, including autonomous driving and IoT, where devices such as wearable devices and vehicles can frequently opt-in and opt-out. Under faulty network connections, prior FL solutions may become fragile when the model parameters fail to be intactly shared among active users due to transmission interruptions. This connection uncertainty is *bidirectional*, which exists either when a participant downloads or uploads parameter updates, leading to nonnegligible information loss of the participating devices. To the best of our knowledge, few pioneer efforts have been made to address the transmission uncertainty in FL, leaving most FL learning schemes at potential risk of undermined performance.

Observing the ***system heterogeneity*** and ***connection uncertainty*** in FL, in this paper, we propose an FL framework that addresses both challenges simultaneously. In our approach, edge devices learn a ***prunable*** neural network by ***self-distillation***, such that a model can be structurally pruned to sub-models that contain adequate knowledge of the learning domain. Towards effective optimization, we

[1]Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA. [2]Department of Electrical and Software Engineering, University of Calgary, Calgary, AB T2N 1N4, Canada.. Correspondence to: Zhuangdi Zhu <zhuzhuan@msu.edu>, Jiayu Zhou <jiayuz@msu.edu>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

propose ***progressive learning***, which articulates the knowledge representation in the model in a nested and progressive structure. This strategy enables FL participants with diverging model architectures to fully devote their knowledge to model aggregation. Furthermore, powered with a ***sequential model transmission*** paradigm, our approach is especially beneficial in amortizing the risk of connection interruptions, since the partially transmitted model parameters before interruption can still contribute self-contained domain knowledge to the recipient.

To the best of our knowledge, we are the first to address both system heterogeneity and connection uncertainty in FL by progressively learning self-distilled networks. Extensive empirical studies have verified that, our proposed approach, dubbed as ***FedResCuE***, is ***Res***ilient to unstable transmission connections while ***C***omm***u***nication ***E***fficient under system heterogeneity, which achieves high asymptotic performance compared with the state-of-the-art.

2. Problem Setting

2.1. Prelimnaries of Federated Learning

Without loss of generality, we consider a learning setting that addresses a representative problem of *multi-class classification*. Let $\boldsymbol{\mathcal{T}} = \{\mathcal{T}_k\}_{k=1}^{K}$ denote the learning domains of edge devices, where a domain $\mathcal{T}_k = \langle \mathcal{X}_k \times \mathcal{Y}_k \rangle$ is defined by a joint input and output distribution. Let $\boldsymbol{\theta}_k$ represent *local* model parameters, and $\boldsymbol{w}$ *global* model parameters, which are usually obtained via parameter-wise averaging on $\{\boldsymbol{\theta}_k\}_{k=1}^{K}$ (McMahan et al., 2017). Denote $\mathcal{L} : \nabla^{\mathcal{Y}} \times \mathcal{Y} \to \mathbb{R}^{+}$ the loss function recgonized by all domains, where $\nabla^{\mathcal{Y}}$ is a *simplex* over $\mathcal{Y}$. The objective of FL is to learn a global model that generalizes well on all devices: $\boldsymbol{w}^{*} = \arg\min_{\boldsymbol{w}} \mathbb{E}_{\mathcal{T}_k \sim \boldsymbol{\mathcal{T}}}[\mathcal{L}(f(\mathcal{X}_k; \boldsymbol{w}), \mathcal{Y}_k)]$, which is approximated by empirical data in practice:

$$\hat{\boldsymbol{w}}^{*} = \arg\min_{\boldsymbol{w}} \frac{1}{K} \sum\nolimits_{k=1}^{K} \left[\frac{1}{n_k} \sum\nolimits_{i=1}^{n_k} \mathcal{L}(f(x_k^i; \boldsymbol{w}), y_k^i) \right],$$

where $\{x_k^i, y_k^i\}_{i=1}^{n_k} \subset \mathcal{T}_k$. A FL system typically involves four iterative phases: i) the *downloading* phase, when the server broadcasts a global model to active users; ii) the *local learning* phase, when active users update their local model parameters; iii) the *uploading* phase, when active users send parameters back to the server, and iv) the *aggregation* phase, when the server derives a global model using user-uploaded parameters.

2.2. Learning with System Heterogeneity

In this paper, we tackle FL under system heterogeneity, where edge devices can learn local models $\boldsymbol{\theta}^k$ with various network architectures due to different capacities in memory and transmission bandwidth (Horvath et al., 2021; Diao et al., 2020). To enable FL with system heterogeneity, participants will first agree on the largest model architecture (*i.e.* a $\times 1$ network), while smaller models in this system are treated as the pruned versions (*i.e.* a $\times p$ network) with a pruning ratio $p < 1$. In Deep Neural Networks (DNNs), such pruning is manifested as reducing the number of channels or filters. For instance, the weight matrix $\boldsymbol{w} \in \mathbb{R}^{m \times n}$ in a Convolution or Linear layer l will be pruned to $\boldsymbol{w}[: m \times p, : n \times p]$ by ratio p. [1]

This strategy leaves one potential drawback to model *aggregation*, in that the global model is obtained via parameter-wise averaging on edge models with heterogeneous sizes. Naively learning and aggregating such model parameters may ignore the divergence in their feature extraction patterns induced by architecture heterogeneity, leading to impaired global model performance. Consequently, the knowledge uploaded by users with diverging model sizes might not be absorbed well by their peers.

2.3. Learning with Unstable Network Connection

Another challenge tackled in this paper is the *connection instability* in FL, which differs from the ***straggler*** issue as discussed in prior art (Reisizadeh et al., 2020; Li et al., 2020). The former refers to *active* users transmitting *partial* instead of complete model parameters due to connection interruption, while the latter results from *inactive* users that did not participate in model learning. Connection interruption may occur bidirectionally either during the downloading phase or the uploading phase. It is a common issue that can be induced by multiple factors, including bandwidth, transmission power, noisy density, and interference (Chen et al., 2019; Nguyen et al., 2021), yet enough effort has been made to address it. A naive solution to connection interruption is to ignore the faulty connected devices and treat them as stragglers (Chen et al., 2020a; Li et al., 2020), which may waste the local learning of those devices that could otherwise be leveraged to improve the global model.

3. Resilient and Communication Efficient FL

Towards addressing the challenges of system heterogeneity and unstable network connections, we aim to learn neural networks that can be *structurally decomposed* for learning, inference, and transmission. Observing the natural property of deep neural networks, we propose to *vertically* decompose a model as a sequence of *columns*, while a column can be one or more model channels described in Section 2.2, depending on the desired granularity. This proposed paradigm enjoys twofold benefits:

- During local learning, the predictive knowledge is *progressively* captured in model columns and can be incrementally enriched with more column connections,

[1] Without losing generality, in this paper, we unify the pruning ratio p for all layers in a model, although such pruning can be extended to choose different ratios p_l for different layers l.

which ***benefits FL with system heterogeneity***, in that the knowledge from heterogeneous models can be structurally aligned in the global model.

- During FL synchronization, model columns are *sequentially* transmitted between the server and the edge device. It makes FL ***resilient to unstable connections***, since losing parts of the tailing columns upon interruption does not lead to a catastrophic undermining of domain representations, and a lightweight submodel that is successfully transmitted still contains intact predictive knowledge. Moreover, this *divided-and-transmit* strategy is also in accord with lower-level transmission protocols. We demonstrate this process in Figure 1.

Orthogonal to our work, there are personalized FL approaches, which either divide a model *horizontally* then transmit the feature extraction layers (Arivazhagan et al., 2019), or selectively transmit parameters with *unordered* structures (Sun et al., 2021). Contrarily, in our FL paradigm, the received columns can be readily assembled for learning and inference. Therefore, instead of discarding the partially received parameters upon connection interruption, they can be effectively utilized for global aggregation or local model initialization.

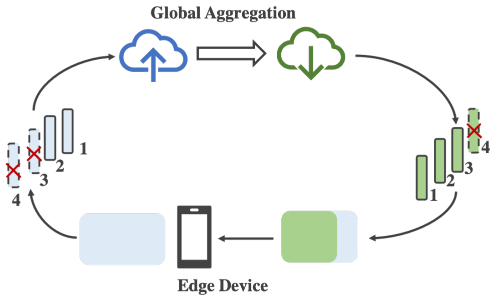


Figure 1: Model parameters are divided and learned as *columns*, which are transmitted *sequentially* between the server and clients, until an interruption occurs to one column, or when all columns are transmitted successfully.

3.1. Learning Self-Distilled Local Models

We aim to learn a model that can be structurally decomposed, arbitrarily *prunable* with reduced *columns*, and dispenses with the need for fine-tuning. We name such a model *self-distilled*, which more concretely, shall satisfy the following objective:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\Theta} \mathcal{L}(f(\mathcal{X};\boldsymbol{\theta}),\mathcal{Y})+ \mathop{\mathbb{E}}_{p\sim\mathcal{P}}[\mathcal{L}(f(\mathcal{X};\boldsymbol{\theta}_{\times p}),\mathcal{Y}) + D_{\mathrm{KL}}[f(\mathcal{X};\boldsymbol{\theta})\|f(\mathcal{X};\boldsymbol{\theta}_{\times p})]], \quad (1)$$

where Θ denotes the parameter space; $\mathcal{X}\times\mathcal{Y}$ is the learning domain; $\mathcal{P} = \{p|p \leq 1, \forall\boldsymbol{\theta} \in \Theta\ f(\mathcal{X};\boldsymbol{\theta}_{\times p}) \subseteq \nabla^{\mathcal{Y}}\}$ is the set of legitimate pruning ratios; $D_{\mathrm{KL}}[p\|q]$ denotes the KL-divergence between distribution p and q.

The notion of *self-distillation* is embodied by two components in Equation 1: The first is $\mathcal{L}(f(\mathcal{X};\boldsymbol{\theta}_{\times p}),\mathcal{Y})$, which induces the largest model $\boldsymbol{\theta}$ to maintain arbitrary submodels $\boldsymbol{\theta}_{\times p}$ that are effective for the learning domain. The other is $D_{\mathrm{KL}}[f(\mathcal{X};\boldsymbol{\theta})\|f(\mathcal{X};\boldsymbol{\theta}_{\times p})]$, in that it encourages the predictive knowledge captured by $\boldsymbol{\theta}$ (teacher), which is manifested as the learned posterior $p(\cdot|\mathcal{X};\boldsymbol{\theta}) \propto f(\mathcal{X};\boldsymbol{\theta})$, to be distilled to the submodel $\boldsymbol{\theta}_{\times p}$ (student) by distribution matching. We verify in Section 5.6 that, the D_{KL} term is especially beneficial when a submodel itself is not representative enough to capture sophisticated features due to a limited number of channels. Hence the posterior distribution from the teacher serves as finer-grained guidance in addition to the label supervision. Moreover, not only is the learned self-distilled network robust against *connection loss*, it also benefits FL under *system heterogeneity*, as the knowledge of an edge model with a larger structure can be adequately conveyed by its submodels, which will be aggregated by the server in the next round and shared with users of smaller model capacities as inductive bias.

3.2. Effective Optimization via Progressive Learning

Optimizing Equation 1 might be prohibitive at first sight, as multiple submodels are bundled within the same network structure, while updating $\boldsymbol{\theta}_{\times p_i}$ may interfere with its nested submodels $\boldsymbol{\theta}_{\times p_j}$ when $p_j < p_i$. Towards effective optimization, we propose an approach that learns a self-distilled model that *incrementally* builds up its feature representation by involving more model *columns*.

Specifically, we first sample a batch of *ordered* pruning ratios $\hat{P} = [p_i|p_i \in \mathcal{P}, p_i < p_{i+1}\ \forall i < S, p_S = 1]_{i=1}^{S}$, then adaptively optimize each sampled submodel towards its objective function. Once a smaller submodel is updated (*e.g.* $\boldsymbol{\theta}_{\times p_1}$), we fix its parameters and update parameters in the subsequent model *columns* (*e.g.* $\boldsymbol{\theta}_{\times p_2}\backslash\boldsymbol{\theta}_{\times p_1}$). This learning scheme leverages the idea of *coordinate descent* (Wright, 2015), which works by successively optimizing one coordinate while fixing the others. As illustrated in Figure 2, predictive knowledge is learned *progressively* by adding more *lateral* connections. More concretely, we update a sampled $\boldsymbol{\theta}_{\times p_i}$ as the following:

$$\boldsymbol{\theta}_{\times p_i} \leftarrow \boldsymbol{\theta}_{\times p_i} - \eta\nabla_{\{\boldsymbol{\theta}_{\times p_i}\backslash\boldsymbol{\theta}_{\times p_{i-1}}\}} J(x;\boldsymbol{\theta}_{\times p_i}), \quad (2)$$

where $J(x;\boldsymbol{\theta}_{\times p_i})$ is the objective function for the current submodel $\boldsymbol{\theta}_{\times p_i}$; η is the learning rate, and $\nabla_{\{\boldsymbol{\theta}_{p_i}\backslash\boldsymbol{\theta}_{\times p_{i-1}}\}}$ denotes the gradients *w.r.t.* parameters that are included in $\boldsymbol{\theta}_{\times p_i}$ but not in $\boldsymbol{\theta}_{\times p_{i-1}}$. In particular, we tailor the objective for each submodel $\boldsymbol{\theta}_{\times p_i}$ as the following:

$$J(x;\boldsymbol{\theta}_{\times p_i}) = \mathcal{L}(f(x;\boldsymbol{\theta}_{\times p_i}),y) + \alpha_i D_{\mathrm{KL}}[f(x;\bar{\boldsymbol{\theta}})\|f(x;\boldsymbol{\theta}_{\times p_i})], \quad (3)$$

where $\bar{\boldsymbol{\theta}}$ is a constant cache of the largest network learned from the last iteration, which serves as the teacher; $\alpha_i := \mathbb{I}[p_i < 1]$ indicates the necessity of knowledge distillation, which renders 0 if the tail of the model columns is sampled (*i.e.* $p_i = 1$), and 1 otherwise. Once all sampled submodels have been visited, we perform an one-time back-propagation to update the teacher. We summarize this model learning approach in Algorithm 1.

Besides being readily prunable, the merits of progressive learning are multifold. First, it accelerates training by adaptively reaching a good *initialization*, which resembles *meta-learning* techniques (Finn et al., 2017). Second, it alleviates the overfitting issue especially when the teacher model structure is surplus given insufficient training data, which we verify in Section 5.3. Furthermore, it also alleviates the permutation-invariant issue in deep neural networks (Wang et al., 2020) by inducing nested and ordered domain representations captured in submodels.

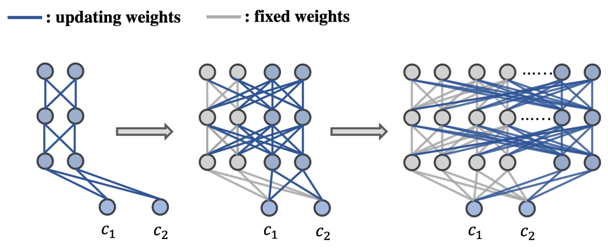


Figure 2: A self-distilled network is learned via progressively updating *columns* of parameters.

Algorithm 1 PROGRESSIVE SELF-DISTILLATION

```
1:  Inputs: Training dataset $D \subset \mathcal{X} \times \mathcal{Y}$; model with parameter set $\boldsymbol{\theta} \in \Theta$; pruning ratios $\mathcal{P}$, learning rate $\eta$, loss function $\mathcal{L}$, constant $S \leq |\mathcal{P}|$.
2:  repeat
3:      Sample batch of $x, y \sim D$.
4:      Sample ordered ratios $\hat{P} = [p_i | p_i \in \mathcal{P}, p_i < p_{i+1} \, \forall i < S, p_S = 1]_{i=1}^{S}$
5:      $\bar{\boldsymbol{\theta}} \leftarrow \text{stop\_gradient}(\boldsymbol{\theta})$, $\boldsymbol{\theta}_{\times 0} = \emptyset$.
6:      for $p_i \sim \hat{P}$ do
7:          $g_i \leftarrow \nabla_{\{\boldsymbol{\theta}_{\times p_i} \setminus \boldsymbol{\theta}_{\times p_{i-1}}\}} J(x; \boldsymbol{\theta}_{\times p_i})$   ▷(Equation 3)
8:          $\boldsymbol{\theta}_{\times p_i} \leftarrow \boldsymbol{\theta}_{\times p_i} - \eta * g_i$.
9:      end for
10:     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta * \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(x; \boldsymbol{\theta}), y)$.
11: until training stop
12: Return $\boldsymbol{\theta}$
```

Case Study on the Effects of Progressive Learning: We illustrate with a preliminary study that, progressively learning a model enjoys the benefits of 1) finding a good initialization for the learning domain and 2) alleviating knowledge forgetting.

In this prototype, we divide a convolution model evenly into two columns and use the first half $\boldsymbol{\theta}_{\times 0.5}$ to learn on a small subset of the MNIST image data. Next, we learn on the SYNTHETIC images with a same data size, by updating $\boldsymbol{\theta}_{\times 1.0} \setminus \boldsymbol{\theta}_{\times 0.5}$ while keeping parameters in $\boldsymbol{\theta}_{\times 0.5}$ fixed. This approach is compared with another variant (*overwriting*), which learns MNIST using $\boldsymbol{\theta}_{\times 0.5}$ then learns SYNTHETIC using the entire model $\boldsymbol{\theta}_{\times 1.0}$. As shown in Table 1, where results are averaged over 6 random seeds, the progressively learned $\times 1.0$ model outperforms its counterpart on both domains. Overwriting $\boldsymbol{\theta}_{\times 0.5}$, on the other hand, may disrupt such initialization brought by learning on the MNIST domain. This also leads to non-negligible forgetting of previously learned representations. See Section A.1 in supplementary for more details.

Accuracy (%) on MNIST and SYNTHETIC.

Domain	Progressive learning	Overwriting
MNIST	**77.95±7.93**	38.17±28.19
SYNTHETIC	**69.48±1.93**	42.30±32.30

Table 1: Progressive *vs.* overwriting learning.

3.3. Proposed Federated Algorithm: *FedResCuE*

Before introducing our FL paradigm, we refine our algorithm with two more techniques to further tackle system heterogeneity and connection instability.

Heterogeneous Model Aggregation: In our FL system, edge users will initially agree on the maximal network architecture and the legitimate pruning ratios $\mathcal{P}$. Next, edge users can choose their maximal local model size with a capacity ratio $p_k \in \mathcal{P}$. During *downloading* (*uploading*) phases, user k with ratio p_k will receive (send) at most $\times p_k$ of model parameters, depending on the network connection. During the *aggregation* phase, active users will only contribute to global parameters that are within their uploading ratios. Accordingly, in the next learning round $t+1$, $\forall \, p_i \in \mathcal{P}$, the global model parameters are derived as follows:

$$\boldsymbol{w}^{t+1}_{\times p_i} \setminus \boldsymbol{w}^{t+1}_{\times p_{i-1}} = \frac{1}{|\mathcal{A}^t_{p_i}|} \sum_{k \in \mathcal{A}^t_{p_i}} \boldsymbol{\theta}^{k,t}_{\times p_i} \setminus \boldsymbol{\theta}^{k,t}_{\times p_{i-1}}, \tag{4}$$

where $\mathcal{A}^t_{p_i} = \{k | k \in \mathcal{A}^t, \, \mathbb{I}[\mathcal{R}(\boldsymbol{\theta}^{k,t}) > p_i]\}$; $\mathcal{A}^t$ denotes the active users from the last round, and $\mathcal{R}(\boldsymbol{\theta}^{k,t})$ is the uploaded network size of user k.

Local Model Padding: Due to unstable network connections, the local device is prone to receiving only partial of the global model during the *downloading* phase. To compensate for the missing global parameters, we leverage the local parameters cached from the last round to *pad* the initialized model to avoid catastrophic information loss. More concretely, at learning round t, we initialize the local model for the k-th user as follows:

$$\boldsymbol{\theta}^{k,t} \leftarrow \boldsymbol{w}^{t}_{\times p^{d,t}_{k}} \cup \{\boldsymbol{\theta}^{k,t-1}_{\times p_k} \setminus \boldsymbol{\theta}^{k,t-1}_{\times p^{d,t}_{k}}\}, \tag{5}$$

where $\boldsymbol{w}^{t}_{\times p^{d,t}_{k}}$ denotes the global parameters downloaded by user, and $p^{d,t}_{k}$ is the downloading ratio, which is possibly smaller than p_k.

Built upon the above techniques, we now present the proposed *FedResCuE* in Algorithm 2. In our algorithm, the workload introduced by progressive learning is lightweight, since the column-wise gradient update has omitted the need for repeated calculation for small submodels, as opposed to some prior arts (Yu et al., 2018). Moreover, as elaborated in Section 5.6.2, *FedResCuE* can obtain superior performance with a small sampling frequency (S). Our approach is also communication efficient, which not only provides flexibility for users to select affordable model architectures but also requires much fewer communication rounds than prior work to reach the predefined performance, as verified in Section 5.5.

Algorithm 2 *FedResCuE*: ***Res***ilient and ***C***omm***u***nication-***E***fficient Federated Learning

```
1: Inputs: Tasks $\{D_k\}_{k=1}^K$; global model $\boldsymbol{w}$ with parameter space $\Theta$; legit pruning ratios $\mathcal{P}$, user capacity ratios $\{p_k\}_{k=1}^K$, models initialized as $\{\boldsymbol{\theta}^{k,0} := \boldsymbol{w}_{\times p_k}\}_{k=1}^K$. learning rate $\eta$, loss function $\mathcal{L}$, sampling frequency $S$, epochs $T$.
2: for $t \in [T]$ do
3:     Aggregate global parameters $\boldsymbol{w}^t$ via Equation 4.
4:     Broadcast $\boldsymbol{w}^t$ to active users $\mathcal{A}^t$.
5:     for user $k \in \mathcal{A}^t$ in parallel do
6:         Download $\boldsymbol{w}^t$ with downloading ratio $p_k^{d,t} \leq p_k$ depending on the connection quality.
7:         $\boldsymbol{\theta}^{k,t} = \boldsymbol{w}^t_{\times p_k^{d,t}} \cup \{\boldsymbol{\theta}^{k,t-1}_{\times p_k} \backslash \boldsymbol{\theta}^{k,t-1}_{\times p_k^{d,t}}\}$ ($\triangleright$ model initialization via Equation 5)
8:         $\boldsymbol{\theta}^{k,t} \leftarrow$ Algorithm 1 $(D_k, \boldsymbol{\theta}^{k,t}, \mathcal{P}, \eta, \mathcal{L}, S)$
9:         Upload $\boldsymbol{\theta}^{k,t}$, with uploading ratio $p_k^{u,t} \leq p_k$ depending on the connection quality.
10:    end for
11: end for
```

4. Related Work

Systematic Heterogeneity in FL is a rising challenge induced by the emergence of FL applications to wireless communications and IoT, where participating devices have varying capacities in computation and transmission. Some work enables heterogeneous model architectures by sharing model predictions on a public dataset instead of sharing model parameters (Taya et al., 2021; Jeong et al., 2018), at the cost of non-negligible performance degradation. Some approaches allow users to share partial model layers, leaving potential opportunities for adopting different architectures for unshared layers (Zhu et al., 2021; Arivazhagan et al., 2019). Another work tackles system heterogeneity by allowing the ensemble of small *base* models on the local device for both training and inference, which loses the possibility of building a deep, wider global model (Hong et al., 2021a). With a different focus on robustness against adversarial attacks, (Hong et al., 2021b) studied the hardware heterogeneity in the case of adversarial federated learning. In general, yet enough efforts have been made to effectively tackle the generalization performance of FL with heterogeneous model architectures, except for a few pioneers such as ***FedHetero*** (Diao et al., 2020) and ***FjORD*** (Horvath et al., 2021), which are extensively analyzed in Section 5.

Network Pruning has long been studied in non-FL scenarios, which aims to prune a lightweight model from a larger one with the maximal knowledge reserved. Prior approaches usually require *fine-tuning* using label supervisions (Han et al., 2016; Lee et al., 2018; Louizos et al., 2018; Dettmers & Zettlemoyer, 2019), Later there emerge *zero-short* pruning (Yu et al., 2018; Cai et al., 2020). Approaches include *structured* pruning that reduces model channels (Ye et al., 2018; Yu et al., 2018; Yu & Huang, 2019; Cai et al., 2020). Orthogonal approaches include *early exit* (Zhang et al., 2021; 2019), which learns *horizontally* pruned networks with reduced number of neural layers. Other pruning strategies follow the lottery ticket hypothesis (Frankle & Carbin, 2018; Ramanujan et al., 2020; Liu et al., 2019). Most prior work derives one submodel after pruning, whereas the ***slimmable*** learning (Yu et al., 2018) and Cai et al. 2020 makes arbitrarily submodels prunable. The idea of prunable models has been applied to FL to tackle system heterogeneity by *FjORD* (Horvath et al., 2021). To the best of our knowledge, we are the first to apply *progressive learning* to address both system heterogeneity and connection instability in FL.

Resilient and Communication Efficient FL addresses FL under restricted or unstable connection bandwidths (Reisizadeh et al., 2020; Gu et al., 2021; Horvath et al., 2021). *FedProx* (Li et al., 2020) tackles *stragglers* via a regularized objective. Other scheduling-based approaches assume that the server has control over the active users (Reisizadeh et al., 2020; Chen et al., 2020a; Nishio & Yonetani, 2019). Not much work has mentioned the faulty connections issues. Some chooses to naively drop the faulty connected edge devices (Chen et al., 2020a; Reisizadeh et al., 2020). Gu et al. 2021 and Yan et al. 2020 suggest to use stale model parameters for the disconnected users. Meanwhile, there are complementary efforts that compress transmitted model parameters by quantization (Alistarh et al., 2017; Amiri et al., 2020) or sketching (Ivkin et al., 2019). Some approaches focus on asynchronous communication (Chen et al., 2020b; Xu et al., 2021). Our algorithm can be potentially combined with related work to further improve resiliency and communication efficiency in FL.

5. Evaluation

In this section, we conduct extensive experiments to answer the following key questions, leaving more experimental details to the supplementary:

1. Is *FedResCuE* resilient to system heterogeneity and unstable network connections?
2. Is *FedResCuE* communication-efficient to reach satisfactory performance with fewer synchronization rounds, compared with the state-of-the-art?
3. Which components of *FedResCuE* have contributed to its *resiliency* and *communication efficiency*?

Results: Experiments below show that *FedResCuE* notably outperforms related work in communication efficiency and asymptotic performance. Its superiority is consistent across different FL settings, and become more prominent under *insufficient* training data, *heterogeneous* model architectures, and *unstable* network connections.

5.1. Experiment Setup

Dataset: We use CIFAR10 and CIFAR100 (Krizhevsky et al., 2009) to simulate edge users with *i.i.d.* data distributions. We also apply DIGITSFIVE (Peng et al., 2019) to simulate users with statistical heterogeneity, which is a multi-domain benchmark with five image datasets: MNIST (LeCun et al., 1998), SVHN (Netzer et al., 2011), USPS (Hull, 1994), Synthetic, and MNIST-M (Ganin & Lempitsky, 2015).

Models: We build a RESNET neural network (He et al., 2016) for learning the CIFAR10 domain, and build a model consisting of 3 CONV2D layers followed by 3 LINEAR layers for learning DIGITSFIVE domains. To enable effective model aggregation under system heterogeneity, we perform careful treatments on the BATCHNORM layers with more details in Section A.3.1 of the supplementary.

Settings of System Heterogeneity: We consider two scenarios to profile the system heterogeneity: 1) the *uniform* setting, where all edge devices maintain the same network architecture, and 2) the *cluster* setting, where the size of local models are randomly sampled from a set $P_C \subset \mathcal{P}$ to represent different system capacity. For the *uniform* setting, one can treat $P_C = \{1.0\}$. For the *cluster* setting, we explored $P_c = \{0.25, 0.5, 0.75, 1.0\}$, and $P_c = \{0.2, 0.35, 0.5, 0.75, 1.0\}$, respectively, to analyze the impacts of varying degrees of system heterogeneity on the learning performance.

Settings of Unstable Communication: A connection error rate ***er*** is generated dynamically to denote the probability that the current column transmission is interrupted. When transmitting one network, we traverse all columns until one column is interrupted based on probability er, or when all columns have been successfully transmitted. Note that the connection loss occurs *bidirectionally*. Hence an edge device may receive or upload models with a smaller size than its assigned architecture. In practice, we set the size of a transmission *column* to be $0.125\times$ of the global network. For experiments that approximate *stable* network connections, we set $er = 0$.

Compared Approaches: In addition to ***FedAvg*** (McMahan et al., 2017), we compare ***FedResCuE*** against the following approaches that tackle system heterogeneity: i) ***FedHetero*** (Diao et al., 2020) extended *FedAvg* to allow edge devices with different model sizes; ii) ***FjORD*** (Horvath et al., 2021) learns prunable local models without *progressive learning*; iii) ***FedSlim*** is a proposed baseline in this paper, in which models are locally updated by following the *slimmable* training (Yu et al., 2018) and globally aggregated as *FedAvg*.

Training: Active users will sync with the server after a complete *epoch* of local training. For faulty connection settings (Section 5.4), the *local padding* strategy is applied to *all* evaluated algorithms for fair comparisons. For the CIFAR10 and CIFAR100 domain, training data is *i.i.d.* sampled and assigned to 20 total users. For the DIGITSFIVE domains, we assign each domain data to 2 unique users. For both types of experiments, 5 active users are randomly selected per communication round. We also evaluate the algorithmic performance given different sizes of training data in Section 5.3.

Evaluation: Unless otherwise specified, the performance is reported using the global model on *all available testing data*, which is evaluated every 2 communication rounds. Results are averaged over 3 random seeds. Asymptotic performance is reported after 300 rounds for CIFAR10, and 100 rounds for DIGITSFIVE.

5.2. Performance Under System Heterogeneity

We apply CIFAR10 data to explore the *uniform* setting, and the *cluster* setting, respectively, as described in Section 5.1.

Results: As shown in Table 2 and Table 3, *FedResCuE* consistently exceeds other approaches in asymptotic performance, especially under a heterogeneous (*cluster*) system. In Table 2, the advantage of *FedResCuE* on the $\times 0.25$ global model demonstrates its benefits to small-capacity users compared to related work, in that *FedResCuE* helps predictive knowledge be distilled from larger models into their nested sub-models, which will be eventually shared by users with smaller model sizes. In the meantime, *FedResCuE* is also more effective than *FjORD* and *FedSlim*, which is largely ascribed to the benefit of its progressive learning, as opposed to a batch-gradient update scheme in prior art.

When the granularity of system heterogeneity increases, as shown in Table 3, *FedResCuE* is still the most advantageous algorithm when evaluated on all of the available sub-models. Moreover, our method potentially can support users with arbitrary legitimate pruning ratios. Note that a

Global Model Accuracy (%) Evaluated on CIFAR10, with Stable Network Connections ($er = 0$).							
Training Data	System Heterogeneity	Evaluated Model	*FedAvg*	*FedHetero*	*FjORD*	*FedSlim*	*FedResCuE*
100%	$P_c = \{1.0\}$ (uniform)	$\boldsymbol{w}_{\times 1}$	81.06±0.63	-	80.57±0.91	81.14±0.76	**81.39±0.20**
		$\boldsymbol{w}_{\times 0.25}$	18.57±0.64	-	69.94±0.65	70.47±0.61	**71.19±0.19**
	$\lvert P_c\rvert = 4$ (cluster)	$\boldsymbol{w}_{\times 1}$	-	76.80±0.53	75.71±0.47	77.49±0.40	**78.22±0.41**
		$\boldsymbol{w}_{\times 0.25}$	-	68.56±0.51	70.98±0.75	73.22±0.34	**73.25±0.47**
20%	$P_c = \{1.0\}$ (uniform)	$\boldsymbol{w}_{\times 1}$	68.03±0.50	-	67.89±1.47	67.96±0.72	**71.27±0.27**
		$\boldsymbol{w}_{\times 0.25}$	16.47±2.24	-	**61.38±1.69**	59.56±1.39	61.12±1.35
	$\lvert P_c\rvert = 4$ (cluster)	$\boldsymbol{w}_{\times 1}$	-	59.38±0.41	62.43±1.65	59.53±0.86	**64.53±1.06**
		$\boldsymbol{w}_{\times 0.25}$	-	55.41±0.39	61.86±1.21	58.31±0.23	**61.98±0.85**

Table 2: Performance in varing degrees of *system heterogeneity* and *training data sufficiency*. We report the best performance from different S for applicable approaches (See Section 5.6.2).

common trend shared by evaluated approaches is that, the middle-sized sub-models, such as $\times 0.75$ model, tend to achieve higher generalization performance than the $\times 1.0$ model under system heterogeneity. We ascribe this phenomenon to the less-frequent update of the $\times 1.0$ model architecture, which is selected by only a small portion of FL users under system heterogeneity.

Global Model Accuracy (%) on CIFAR10, with $\lvert P_C\rvert = 5$					
Cluster ratio p	$\times 0.2$	$\times 0.35$	$\times 0.5$	$\times 0.75$	$\times 1.0$
FedResCuE	**57.98**	**62.59**	**63.53**	**62.53**	**61.99**
FjORD	56.24	59.25	59.89	59.23	58.74
FedHetero	52.84	57.17	58.58	57.94	57.57
FedSlim	55.19	57.51	57.84	56.87	56.35

Table 3: Given 5 *clusters* and 20% training data, *FedResCuE* outperforms its counterparts that support system heterogeneity on all pruned sub-models.

5.3. Performance Given Insufficient Training Data

To analyze the impacts of data sufficiency, we assign 100% and 20% of the CIFAR10 to users for training, respectively, assuming a stable network connection ($er = 0$).

Results: As shown in Table 2, when training data is sufficient with *i.i.d.* distributions, all algorithms perform comparably well. However, given only 20% of the training data, both *FjORD* and *FedSlim* slightly underperform *FedAvg* when evaluated using the $\times 1.0$ model, while *FedResCuE* remarkably outperforms all others. In fact, the progressive parameter update in *FedResCuE* can make a subnetwork a good *initialization* for the encompassing larger submodel, which is analogous to meta-learning that delivers an effective model with fewer shots of training. This is especially beneficial in the lack of training data. Contrarily, *FjORD* and *FedSlim*, which adopt batch-gradient updates for learning prunable models, may struggle with the interference of noisy gradients, which can be further amplified in a potentially overfit model.

5.4. Performance Under Unstable Connections

We apply both CIFAR10 and DIGITSFIVE to explore the scenario with unstable connections. For experiments on the CIFAR10 domain, er is dynamically sampled, with $er \sim [0.1, 0.2]$. For the DIGITSFIVE domains, er is set to be a constant depending on the specific domain.

Results: Under faulty network connections, *FedResCuE* consistently outperforms other approaches under both *i.i.d.* and heterogeneous data distributions. Given the CIFAR10 domain (Table 4), *FedResCuE* achieves higher accuracy than *FedSlim* and *FjORD* with a significant margin, which we ascribe to both its progressive learning procedure and the proposed optimization objective. In fact, given unstable connections, a smaller network architecture turns out to be more reliable, whose transmission is less likely to be interrupted. *FedResCuE* can facilitate FL in this scenario, as it follows a progressive scheme to gradually learn the larger network, which captures the complementary representation built upon its nested smaller submodels. On the other hand, both *FedResCuE* and *FjORD* are more competent than *FedSlim*, which indicates that solely performing distillation from the teacher ($\times 1.0$ model) to student (submodel), as *FedSlim* performs, may be insufficient to deliver reliable submodels, especially given a model with information staleness caused by transmission loss. Contrarily, the optimization of *FedResCuE* (Equation 1), which encourages both knowledge distillation and submodel-learning with label supervision, is a more robust strategy.

When unstable connection is bundled with system heterogeneity: We found that enabling system heterogeneity in FL naturally introduces resiliency to connection instability, which can be revealed by the performance gain of *FedHetero* over *FedAvg* in Table 4. In fact, *FedHetero* can be treated as *macro*-level prunable training, although the resiliency brought by diversified model architectures is less effective than learning self-distilled networks.

We also explore on the CIFAR100 domain to verify the ef-

Global Model Accuracy (%) Evaluated on CIFAR10 Under Connection Loss ($er > 0$).						
System Heterogeneity	Evaluated Model	*FedAvg*	*FedHetero*	*FjORD*	*FedSlim*	***FedResCuE***
$P_c = \{1.0\}$ (uniform)	$\boldsymbol{w}_{\times 1}$	50.36±2.17	-	61.79±1.62	57.31±1.27	**70.02±0.40**
	$\boldsymbol{w}_{\times 0.25}$	12.58±0.51	-	60.20±1.67	55.33±0.89	**67.40±0.84**
$\lvert P_c\rvert = 4$ (cluster)	$\boldsymbol{w}_{\times 1}$	-	60.92±1.33	64.52±0.60	62.35±1.76	**69.78±0.74**
	$\boldsymbol{w}_{\times 0.25}$	-	59.70±0.64	64.11±0.41	61.77±1.62	**68.83±1.00**

Table 4: Performance under *unstable network connections*, given 100% of training data, and $0.1 \leq er \leq 0.2$.

fectiveness of our approach. In Figure 3, where we adopted a WIDERES model architecture (Zagoruyko & Komodakis, 2016) for the $\times 1.0$ model, with $|P_c| = 5$, and $er = 0.1$. *FedResCuE* surpassed all baselines on each of the pruned sub models. Compared with *FjORD* and *FedHetero* the superiority of our approach is prominent especially given smaller pruning ratios.

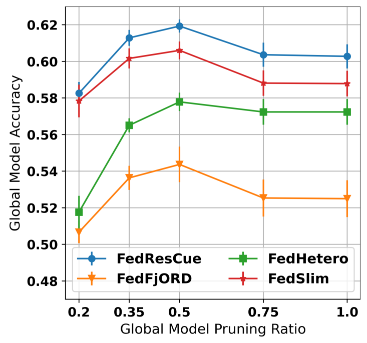


Figure 3: Pruned model accuracy on CIFAR100, given 100% training data, $|P_c| = 5$ and $0.05 \leq er \leq 0.1$.

When unstable connection is bundled with data heterogeneity: As shown in Table 5, under domain-dependent connection errors, *FedResCuE* also shows consistent robustness against heterogeneous statistical distributions. Note that a small training dataset from DIGITSFIVE is applied to ensure the necessity of FL. Hence *Local* learning without sharing parameters yields worse performance than FL. The performance gain of *FedResCuE* resides in both the small ($\times 0.25$) and the large ($\times 1.0$) model (See Section A.3.4 in supplementary). Contrarily, *FjORD* and *FedSlim* may underperform *FedAvg* when evaluated using the $\times 1.0$ model, indicating their potential drawback given insufficient data, which we investigate more in Section 5.3.

5.5. Evaluation of Communication Efficiency

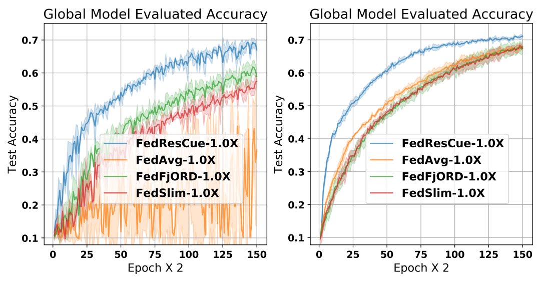
We analyze the communication efficiency via the number of FL synchronization rounds for the global model to reach a reasonable performance. As shown in Table 6, under the *cluster* setting with $\{P_c\} = 4$, *FedResCuE* constantly learns faster to obtain a predefined accuracy, requiring fewer communication rounds than all its peers. The communication efficiency of *FedResCuE* also resides in its flexibility in user model architectures, in that devices that choose a small model architecture can be further benefited by transmitting fewer parameters per communication round. Although other baselines *e.g. FedHetero* also enable system heterogeneity, they require non-negligible more communication rounds to perform comparably to *FedResCuE*. Accompanying performance curves of the $\times 1.0$ model are visualized in Figure 4 (See Section A.3.2 in supplementary for comprehensive results).

Accuracy(%) on DIGITSFIVE, Given 5% Training Data.					
Domain	SVHN	Syn	USPS	MNIST	MNIST-M
Local	45.88	62.04	89.95	85.84	61.99
FedAvg	**69.54**	80.17	94.38	93.61	75.22
FedSlim	66.77	78.95	94.00	93.80	73.95
FjORD	49.72	57.12	68.60	68.01	56.29
FedResCuE	69.32	**80.57**	**95.17**	**95.05**	**76.89**

Table 5: *FedResCuE* is the most robust algorithm given heterogeneous data and domain-dependent connection error.

Communication Efficiency on CIFAR10 dataset.					
Acc	Model Size	*FedHetero*	*FjORD*	*FedSlim*	*FedResCuE*
		100 % training data, $0.1 \leq er \leq 0.2$.			
60%	$\boldsymbol{w}_{\times 0.5}$	256.7	218.0	253.3	**124.7**
		20 % training data, $er = 0$			
55%	$\boldsymbol{w}_{\times 0.5}$	180.7	156.0	192.0	**96.0**

Table 6: *FedResCuE* requires notably fewer communication rounds to reach the predefined accuracy (Acc).



(a) $0.1 \leq er \leq 0.2$ (b) 20% training data

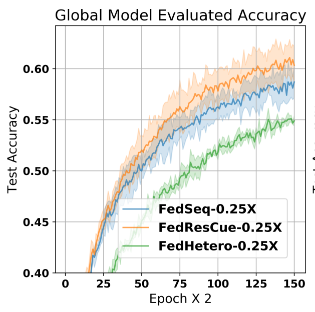
Figure 4: Learning curves evaluated on the $\times 1.0$ model.

5.6. Sensitivity Analysis

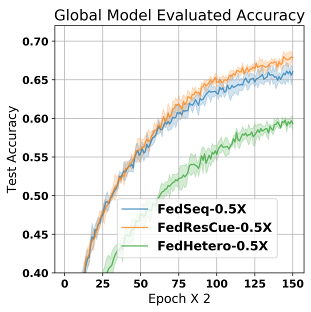
5.6.1. EFFECTS OF KNOWLEDGE DISTILLATION:

To analyze the role of knowledge distillation in our model learning, we design a variant of our approach called *FedSeq* to compare against *FedResCuE*. Particularly, the optimization objective of *FedSeq* does not require minimizing the KL-divergence between the teacher $\bar{\boldsymbol{\theta}}$ and a student $\boldsymbol{\theta}_{\times p_i}$, *i.e.* it always sets the term α_i to 0 in Equation 3.

Results: Knowledge distillation is especially beneficial to smaller submodels, whereas the gap between *FedSeq* and *FedResCuE* gradually diminishes when evaluating using larger submodels. As illustrated in Figure 5, where 20% of the CIFAR10 training data is given, both approaches are learning comparably well in initial training stages, while *FedResCuE* converges to higher asymptotic performance. The learning curves demonstrate that it is beneficial to distill representation knowledge from a large, complete model to smaller submodels, in that the larger model has more channels to capture refined domain knowledge. See Section A.4.1 in supplementary for more results.



(a) $\times_{0.25}$ model.



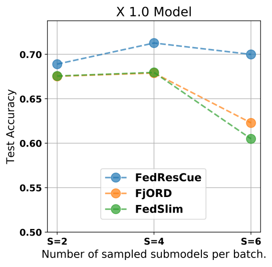
(b) $\times_{0.5}$ model.

Figure 5: Knowledge-distillation in *FedResCuE* can benefit smaller sub-models, compared to its ablated variants.

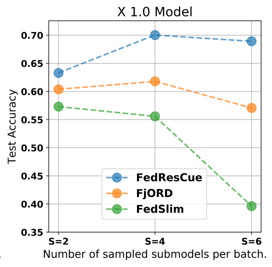
5.6.2. Impacts of Submodel Sampling:

Sampling frequency, denoted as $S = |\hat{P}|$ in Algorithm 1, is the number of submodels sampled per batch update. A key question regarding *FedResCuE* is: *how does S affect the learning performance?* This question is equally intriguing to *FedSlim* and *FjORD*, both of which require submodel sampling. To answer this question, we traverse different choices of S, while the sampling granularity is set to be $\times 0.05$ of the largest model width.

Results: As shown in Figure 6, *FedResCuE* is constantly the most robust under different S. In the meantime, a moderate number of ratio sampling (*e.g.* $S = 4$) benefit most evaluated algorithms, while oversampling with a large S causes non-negligible performance degradation on *FedSlim* and *FjORD*. Their over-sensitivity to S can be induced by their batch-gradient update scheme, which may cause the gradients *w.r.t.* larger submodels to interfere with those of smaller ones when aggregating all gradients in one batch, hence undermining model performance. On the contrary, *FedResCuE* alleviates such issues by using a progressive learning scheme that decouples such mutual impacts.



(a) 20% training data.



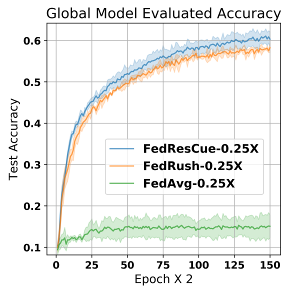
(b) $0.1 \leq er \leq 0.2$.

Figure 6: Impacts of submodel sampling frequency.

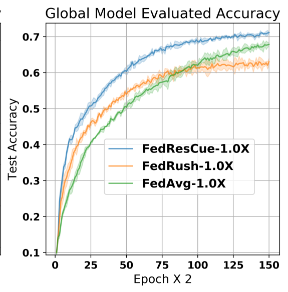
5.6.3. Effects of Progressive Learning:

To evaluate the efficacy of the progressive learning scheme, we compare *FedResCuE* against an intuitive alternative named *FedRush*, which directly updates the sampled submodel without freezing the preceding parameters.

Results: As shown in Figure 7, a rush gradient update as in *FedRush* leads to notably undermined performance. Particularly, when updating the $\boldsymbol{\theta}_{\times p_{i+1}}$ model, *FedRush* overwrites the parameters in $\boldsymbol{\theta}_{\times p_i}$, which could otherwise serve as a good initialization to support the subsequent submodels. Contrarily, *FedResCuE* learns more effectively by avoiding the potential information forgetting.



(a) $\times_{0.25}$ model.



(b) $\times_{1.0}$ model.

Figure 7: Progressive learning in *FedResCuE* guarantees a high performance for the $\times 1.0$ model.

6. Conclusion

We proposed *FedResCuE* to address both system heterogeneity and unstable network connections, by learning *self-distilled* networks in a *progressive* manner, which proves to be communication-efficient with higher performance in the proposed Federated Learning settings compared with the state-of-the-art.

7. Acknowledgement

This research was supported by the National Institute on Aging 1RF1AG072449, the Office of Naval Research N00014-20-1-2382, and the National Science Foundation IIS-1749940.

References

Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in Neural Information Processing Systems*, 2017.

Amiri, M. M., Gunduz, D., Kulkarni, S. R., and Poor, H. V. Federated learning with quantized global model updates. *arXiv preprint arXiv:2006.10672*, 2020.

Arivazhagan, M. G., Aggarwal, V., Singh, A. K., and Choudhary, S. Federated learning with personalization layers. *arXiv preprint arXiv:1912.00818*, 2019.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2020.

Chen, M., Yang, Z., Saad, W., Yin, C., Poor, H. V., and Cui, S. Performance optimization of federated learning over wireless networks. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6. IEEE, 2019.

Chen, M., Yang, Z., Saad, W., Yin, C., Poor, H. V., and Cui, S. A joint learning and communications framework for federated learning over wireless networks. *IEEE Transactions on Wireless Communications*, 20(1):269–283, 2020a.

Chen, Y., Ning, Y., Slawski, M., and Rangwala, H. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, pp. 15–24. IEEE, 2020b.

Dettmers, T. and Zettlemoyer, L. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Diao, E., Ding, J., and Tarokh, V. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264*, 2020.

Dinh, C. T., Tran, N. H., and Nguyen, T. D. Personalized federated learning with moreau envelopes. *34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Du, Z., Wu, C., Yoshinaga, T., Yau, K.-L. A., Ji, Y., and Li, J. Federated learning for vehicular internet of things: Recent advances and open issues. *IEEE Open Journal of the Computer Society*, 1:45–61, 2020.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pp. 1126–1135. PMLR, 2017.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Ganin, Y. and Lempitsky, V. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pp. 1180–1189. PMLR, 2015.

Gu, X., Huang, K., Zhang, J., and Huang, L. Fast federated learning in the presence of arbitrary device unavailability. *35th Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., and Ramage, D. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hong, J., Wang, H., Wang, Z., and Zhou, J. Efficient split-mix federated learning for on-demand and in-situ customization. In *International Conference on Learning Representations*, 2021a.

Hong, J., Wang, H., Wang, Z., and Zhou, J. Federated robustness propagation: Sharing adversarial robustness in federated learning. *arXiv preprint arXiv:2106.10196*, 2021b.

Hong, J., Zhu, Z., Yu, S., Wang, Z., Dodge, H. H., and Zhou, J. Federated adversarial debiasing for fair and transferable representations. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 617–627, 2021c.

Horvath, S., Laskaridis, S., Almeida, M., Leontiadis, I., Venieris, S. I., and Lane, N. D. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *35th Conference on Neural Information Processing Systems (NeurIPS).*, 2021.

Hull, J. J. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.

Ivkin, N., Rothchild, D., Ullah, E., Braverman, V., Stoica, I., and Arora, R. Communication-efficient distributed sgd with sketching. *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.

Jeong, E., Oh, S., Kim, H., Park, J., Bennis, M., and Kim, S.-L. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *arXiv preprint arXiv:1811.11479*, 2018.

Kanchi, S., Sandilya, S., Bhosale, D., Pitkar, A., and Gondhalekar, M. Overview of lte-a technology. In *2013 IEEE global high tech congress on electronics*, pp. 195–200. IEEE, 2013.

Khan, L. U., Saad, W., Han, Z., Hossain, E., and Hong, C. S. Federated learning for internet of things: Recent advances, taxonomy, and open challenges. *IEEE Communications Surveys & Tutorials*, 2021.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

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

Lee, N., Ajanthan, T., and Torr, P. H. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450, 2020.

Li, X., Jiang, M., Zhang, X., Kamp, M., and Dou, Q. Fedbn: Federated learning on non-iid features via local batch normalization. *ICLR*, 2021.

Liu, Y., Huang, A., Luo, Y., Huang, H., Liu, Y., Chen, Y., Feng, L., Chen, T., Yu, H., and Yang, Q. Fedvision: An online visual object detection platform powered by federated learning. In *AAAI*, 2020.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *ICLR*, 2019.

Louizos, C., Welling, M., and Kingma, D. P. Learning sparse neural networks through l_0 regularization. *ICLR*, 2018.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. 2011.

Nguyen, K., Drew, S., Huang, C., and Zhou, J. Edgepv: collaborative edge computing framework for task offloading. In *ICC 2021-IEEE International Conference on Communications*, pp. 1–6. IEEE, 2021.

Nishio, T. and Yonetani, R. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pp. 1–7. IEEE, 2019.

Peng, X., Bai, Q., Xia, X., Huang, Z., Saenko, K., and Wang, B. Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1406–1415, 2019.

Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., and Rastegari, M. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11893–11902, 2020.

Raza, S., Liu, W., Ahmed, M., Anwar, M. R., Mirza, M. A., Sun, Q., and Wang, S. An efficient task offloading scheme in vehicular edge computing. *Journal of Cloud Computing*, 9:1–14, 2020.

Reisizadeh, A., Tziotis, I., Hassani, H., Mokhtari, A., and Pedarsani, R. Straggler-resilient federated learning: Leveraging the interplay between statistical accuracy and system heterogeneity. *arXiv preprint arXiv:2012.14453*, 2020.

Rieke, N., Hancox, J., Li, W., Milletari, F., Roth, H. R., Albarqouni, S., Bakas, S., Galtier, M. N., Landman, B. A., Maier-Hein, K., et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1):1–7, 2020.

Sun, B., Huo, H., Yang, Y., and Bai, B. Partialfed: Cross-domain personalized federated learning via partial initialization. *Advances in Neural Information Processing Systems*, 34, 2021.

Taya, A., Nishio, T., Morikura, M., and Yamamoto, K. Decentralized and model-free federated learning: Consensus-based distillation in function space. *arXiv preprint arXiv:2104.00352*, 2021.

Wang, H., Li, X., Ji, H., and Zhang, H. Federated offloading scheme to minimize latency in mec-enabled vehicular networks. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6. IEEE, 2018.

Wang, H., Yurochkin, M., Sun, Y., Papailiopoulos, D., and Khazaeni, Y. Federated learning with matched averaging. *ICLR*, 2020.

Wright, S. J. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

Xu, C., Qu, Y., Xiang, Y., and Gao, L. Asynchronous federated learning on heterogeneous devices: A survey. *arXiv preprint arXiv:2109.04269*, 2021.

Yan, Y., Niu, C., Ding, Y., Zheng, Z., Wu, F., Chen, G., Tang, S., and Wu, Z. Distributed non-convex optimization with sublinear speedup under intermittent client availability. *arXiv preprint arXiv:2002.07399*, 2020.

Ye, J., Lu, X., Lin, Z., and Wang, J. Z. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018.

Yu, J. and Huang, T. S. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1803–1811, 2019.

Yu, J., Yang, L., Xu, N., Yang, J., and Huang, T. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zhang, L., Song, J., Gao, A., Chen, J., Bao, C., and Ma, K. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3713–3722, 2019.

Zhang, L., Bao, C., and Ma, K. Self-distillation: Towards efficient and compact neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Zhu, Z., Hong, J., and Zhou, J. Data-free knowledge distillation for heterogeneous federated learning. *International Conference on Machine Learning*, 2021.

A. Appendix

A.1. Case Study: Effects of Progressive Learning

For this prototype experiment, we apply the same network architecture used for learning DIGITSFIVE domains, which is presented in Section A.6.1. We illustrate the *progressive* and *overwriting* learning approaches in Figure 8 and Figure 9, respectively. For progressive learning, parameters in $\boldsymbol{\theta}_{\times 0.5}$ are updated first for learning the MNIST domain, which are then frozen when updating parameters in $\boldsymbol{\theta}_{\times 1.0} \backslash \boldsymbol{\theta}_{\times 0.5}$ for learning the SYNTHETIC domain. Contrarily, the overwriting approach updates the entire set of model parameters when learning on the SYNTHETIC domain. We set the learning rate to 0.01 and train 10 epochs for learning each domain, with a batch size set to be 32. For the 10% (5%) training data setting, we apply 743 (371) training samples for learning both MNIST and SYNTHETIC domains. Experimental results are averaged from 6 random seeds, with seed numbers set to be 1, 3, 5, 7, 9, 11, respectively.

As shown in Table 7 and Table 8, the overwriting learning procedure leads to drastic information forgetting on the previously learned domain (MNIST). On the contrary, evaluations on both the $\times 0.5$ model and $\times 1.0$ model indicate that a *progressive* learning scheme can adaptively build up knowledge on the new training data without undermining the preceding knowledge representations. In fact, the new collateral connections that are progressively learned can also improve the overall model performance on the MNIST domain. For instance, when training using 5% of domain data, model performance on the MNIST domain has been improved from 66.46% (on the $\times 0.5$ model) to 72.27% (on the $\times 1.0$ model), which verifies the advantage of our progressive model learning strategy.

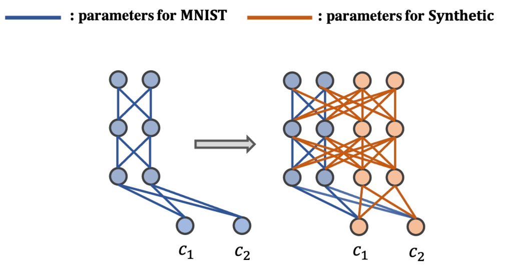


Figure 8: Illustration of ***progressive*** learning.

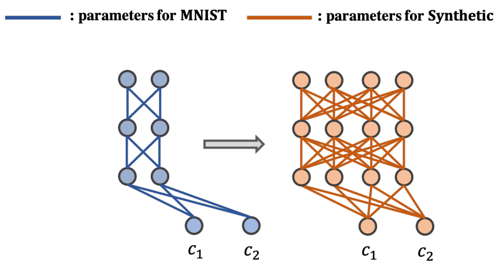


Figure 9: Illustration of ***overwriting*** learning.

Accuracy (%) Evaluated on the × **1.0** Model		
Domain	Progressive Learning	Overwriting
	Given 10 % training data	
MNIST	**77.95±7.93**	38.17±28.19
SYNTHETIC	**69.48±1.93**	42.30±32.30
	Given 5 % training data	
MNIST	**72.27±9.43**	19.60±17.53
SYNTHETIC	**52.43±5.88**	19.03±16.32

Table 7: Progressive *vs.* overwriting learning.

Accuracy (%) on **MNIST** Evaluated on the × **0.5** Model		
Training data	Progressive Learning	Overwriting
	Given 10 % training data	
10%	**74.99±11.26**	33.51±24.95
	Given 5 % training data	
5%	**66.46±14.20.**	19.23±16.21

Table 8: The *overwriting* learning approach leads to severe forgetting on previously learned knowledge.

A.2. Details of Evaluated Related Work

One related approach to learn prunable models is the ***slimmable*** network proposed in (Yu et al., 2018), which works by sampling a set of pruning ratios $P = \{p_i | 0 < p_i \leq 1\}_{i=1}^{|P|}$ then performing a *batch gradient* update, as summarized in Algorithm 3. During each learning iteration, a constant copy of the complete network $\bar{\boldsymbol{\theta}}$ is referred to as *teacher*, whose prediction distribution $p(\cdot|x; \bar{\boldsymbol{\theta}}) \propto f(x; \boldsymbol{\theta})$ is distilled into the submodel $\boldsymbol{\theta}_{\times p_i}$, *i.e.* the *student*. The gradients for both teacher and student models are later aggregated to update the network parameter.

Slimmable learning is proposed for non-FL settings, where sufficient data is accessible on a central machine. There are potential drawbacks of directly applying it to our FL setting. First is the error propagation issue: when the teacher model is underperforming *e.g.* given insufficient training or connection loss, its sub-optimality may be propagated back into the student. In contrast, we propose an objective (in Equation 1) that alleviates this issue by introducing $\min_{\boldsymbol{\theta}_{\times p}} \mathcal{L}(f(\mathcal{X}; \boldsymbol{\theta}_{\times p}), \mathcal{Y})$, which allows submodels to receive label supervisions. Second, the gradients for different submodels may *interfere* with each other when being aggregated, which weakens the model's learning effect, as verified in Section 5.6.2.

Another compared related work is ***FjORD*** (Horvath et al., 2021), which also learns prunable networks for a heterogeneous FL system. We summarize its local model updating procedure in Algorithm 4. In practice, we also apply loss backpropagation through the teacher model (*i.e.* line 9 in Algorithm 4) when optimizing towards the *FjORD* objectives, which is suggested in (Horvath et al., 2021) to further improve their model performance. Note that *FjORD* does not involve progressive parameter updates, the effects of which will be elaborated more in Section A.4.1. In our experiments, we explore different choices of S and perform evaluations on *FjORD* and *FedSlim* using their optimal S accordingly.

Algorithm 3 SLIMMABLE-TRAINING ((Yu et al., 2018))

```
1: Inputs: training dataset $D \subset \mathcal{X} \times \mathcal{Y}$; model with parameter set $\boldsymbol{\theta}$; pruning ratios $\mathcal{P}$, learning rate $\eta$, loss function $\mathcal{L}$,
   constant $S \leq |\mathcal{P}|$.
2: repeat
3:     Sample batch $x, y \sim D$.
4:     Sample widths $\hat{P} = [p_i | p_i \sim \mathcal{P}]_{i=1}^{S}$.
5:     $\bar{\boldsymbol{\theta}} \leftarrow$ stop_gradient($\boldsymbol{\theta}$).
6:     for $p_i \sim \hat{P}$ do
7:         $g_i \leftarrow \nabla_{\boldsymbol{\theta}_{\times p_i}} KL[f(x; \bar{\boldsymbol{\theta}}) \| f(x; \boldsymbol{\theta}_{\times p_i})]$.
8:     end for
9:     $g_{\boldsymbol{\theta}} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(x; \boldsymbol{\theta}), y)$
10:    Update parameter $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta * (g_{\boldsymbol{\theta}} + \sum_{i \in |S|} g_i)$
11: until training stop
```

Algorithm 4 Local Model Update for *FjORD*((Horvath et al., 2021))

```
1: Inputs: training dataset $D \subset \mathcal{X} \times \mathcal{Y}$; model with parameter set $\boldsymbol{\theta}$; pruning ratios $\mathcal{P}$, learning rate $\eta$, loss function $\mathcal{L}$,
   constant $S \leq |\mathcal{P}|$.
2: repeat
3:     Sample batch $x, y \sim D$.
4:     Sample widths $\hat{P} = [p_i | p_i \sim \mathcal{P}]_{i=1}^{S}$.   ▷ ($S = 1$ in (Horvath et al., 2021))
5:     $\bar{\boldsymbol{\theta}} \leftarrow$ stop_gradient($\boldsymbol{\theta}$).
6:     for $p_i \sim \hat{P}$ do
7:         $g_i \leftarrow \nabla_{\boldsymbol{\theta}_{\times p_i}} \left\{\mathcal{L}(f(\mathcal{X}; \boldsymbol{\theta}_{\times p_i}), \mathcal{Y}) + D_{\mathrm{KL}}[f(x; \bar{\boldsymbol{\theta}}) \| f(x; \boldsymbol{\theta}_{\times p_i})]\right\}$.
8:     end for
9:     $g_{\boldsymbol{\theta}} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(x; \boldsymbol{\theta}), y)$
10:    Update parameter $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta * (g_{\boldsymbol{\theta}} + \sum_{i \in |S|} g_i)$
11: until training stop
```

A.3. Experiments

A.3.1. TREATMENTS ON BATCH NORMALIZATION LAYERS

The BATCHNORM layers need to be carefully tackled for effectively training prunable models. Prior arts either make the BATCHNORM layers localized on user devices to improve personalized FL, such as proposed in *FedBN*(Li et al., 2021), or learn individual BATCHNORM modules for each possible submodel (Yu et al., 2018), which unnecessarily inreases the number of learnable parameters. In our work, we apply a lightweight approach that still shares the trainable parameters in BATCHNORM layers, while disabling tracking the running average and variance of training batches, which proves to be an effective scheme across different datasets (Diao et al., 2020). For fair comparisons, we apply the same strategy on all baselines, including the *FedAvg*.

Table 9 summarizes the impacts of different practices regarding BATCHNORM layers on the *FedAvg* performance, which indicates that personalizing BN layers, as *FedBN* applies, might not be good practice for learning *i.i.d.* yet complicated domains such as CIFAR10. On the contrary, decoupling feature learning from relying on tracking the mean and variance of training data, proves to be effective on both *FedAvg* and our proposed self-distillation approach.

Effects of Different BatchNorm Layer Configurations.				
Algorithm	Personalized BN Layers	Tracking Training Status	Test Accuracy (%) (Given 100% training)	Test Accuracy (%) (Given 20% training)
*FedAvg**	×	×	81.06±0.63	68.03±0.50
FedAvg	×	✓	79.73±0.22	68.14±0.47
FedBN	✓	✓	76.12±0.74	60.41±0.57

Table 9: We adopted *FedAvg** as the baseline implementation in the main paper.

A.3.2. Overview of Communication Efficiency

We summarize the communication efficiency of evaluated algorithms on Cifar10 domain in Table 10, where evaluation is performed under a heterogeneous FL system. Results demonstrate that *FedResCuE* requires the least communication rounds to reach the predefined accuracy.

Communication Efficiency on Cifar10 Dataset, *Cluster* Setting					
Accuracy	Model Size	*FedHetero*	*FjORD**	*FedSlim**	*FedResCuE*
	***100 $\%$ training data,* $0.1 \leq er \leq 0.2$.**				
65%	$\boldsymbol{w}_{\times 1}$	-	-	-	**174.7±22.2**
60%	$\boldsymbol{w}_{\times 0.5}$	256.7±19.3	218.0±5.9	253.3±26.5	**124.7±11.1**
55%	$\boldsymbol{w}_{\times 0.25}$	222.7±6.8	165.3±5.2	190.7±23.8	**85.3±1.9**
	***20 $\%$ training data,* $er = 0$**				
60%	$\boldsymbol{w}_{\times 1}$	-	244.0±33.5	188.7±133.4	**166.0±14.2**
55%	$\boldsymbol{w}_{\times 0.5}$	180.7±14.8	156.0±13.4	192.0±7.1	**96.0±2.8**
50%	$\boldsymbol{w}_{\times 0.25}$	163.3±11.5	122.7±4.1	168.7±5.2	**76.7±1.9**

Table 10: Communication Efficiency Overview. '-' indicates that predefined performance is not reached before training ends.

A.3.3. Performance with System Heterogeneity

In addition to the discussion in Section 5.2 of the main paper, we provide two more observations regarding system heterogeneity: 1) when FL is free from the risk of a connection interruption, a *uniform* setting in which the same model architecture is assigned to all the edge devices, is generally more beneficial than a *cluster* setting, where heterogeneous and smaller models are enabled. This result conforms to our perception that larger models can capture more representative feature maps for the learning domain, while smaller models sacrifice such information gain for computation and communication efficiency. 2) Contrarily, the diversity in model architecture makes FL resilient to connection loss to some extent. Specifically, performing parameter-wise averaging on heterogeneous models, just as *FedHetero* applies, can potentially make submodels within the global model function as well, in that the submodel parameters are contributed by smaller-capacity users. Therefore, it can outperform *FedAvg* under unpredictable connection losses, although such an advantage is much less effective than *FedResCuE* with self-distillation learning. We provide a more comprehensive summary in Table 11, with the accompanying learning curves illustrated in Section A.5.

A.3.4. Performance Under Connection Loss

In our experiments, we simulate the unpredictable transmission scenarios with connection loss via a random variable er, which denotes the probability that the current *column* transmission is interrupted. For experiments on the Cifar10 domain, er is first *uniformly* sampled within an error bound (*e.g.* $er \in [0.1, 0.2]$). Next, we use er to determine whether the current column transmission will be interrupted, by comparing it against another uniformly-sampled random variable ϵ, which leads to interruption *iff* $\epsilon < er$. We perform such random sampling on er and ϵ for transmitting each column in a model, while a column for transmission is set to be $\times 0.125$ of the global model width.

Performance overview on the Cifar10 domain given different connection loss rates is presented in Table 12. For experiments on the DigitsFive domains, we set constant er that depends on the domain type. Performance on all five domains is provided in Table 13. Note that we applied a small training dataset from DigitsFive to ensure the necessity of FL, so that *Local* learning without parameter sharing yields worse performance than FL.

Global Model Accuracy (%) Evaluated on CIFAR10.							
Training Data	User Capacity	Evaluated Model	*FedAvg*	*FedHetero*	*FjORD**	*FedSlim**	*FedResCuE*
100%	$\forall k\ p_k = 1$ **(uniform)**	$\boldsymbol{w}_{\times 1}$	81.06±0.63	-	80.57±0.91	81.14±0.76	**81.39±0.20**
		$\boldsymbol{w}_{\times 0.5}$	49.46±1.10	-	77.59±0.31	77.47±0.75	**77.82±0.15**
		$\boldsymbol{w}_{\times 0.25}$	18.57±0.64	-	69.94±0.65	70.47±0.61	**71.19±0.19**
	$\forall k\ p_k \sim P_C$ **(cluster)**	$\boldsymbol{w}_{\times 1}$	-	76.80±0.53	75.71±0.47	77.49±0.40	**78.22±0.41**
		$\boldsymbol{w}_{\times 0.5}$	-	76.08±0.50	75.35±0.43	77.12±0.40	**77.58±0.33**
		$\boldsymbol{w}_{\times 0.25}$	-	68.56±0.51	70.98±0.75	73.22±0.34	**73.25±0.47**
20%	$\forall k\ p_k = 1$ **(uniform)**	$\boldsymbol{w}_{\times 1}$	68.03±0.50	-	67.89±1.47	67.96±0.72	**71.27±0.27**
		$\boldsymbol{w}_{\times 0.5}$	36.56±1.74	-	65.13±1.70	64.80±1.19	**68.15±0.39**
		$\boldsymbol{w}_{\times 0.25}$	16.47±2.24	-	**61.38±1.69**	59.56±1.39	61.12±1.35
	$\forall k\ p_k \sim P_C$ **(cluster)**	$\boldsymbol{w}_{\times 1}$	-	59.38±0.41	62.43±1.65	59.53±0.86	**64.53±1.06**
		$\boldsymbol{w}_{\times 0.5}$	-	59.95±0.36	63.06±1.61	60.36±0.45	**66.37±0.78**
		$\boldsymbol{w}_{\times 0.25}$	-	55.41±0.39	61.86±1.21	58.31±0.23	**61.98±0.85**

Table 11: FL Performance with *i.i.d.* user statistical distributions.

Global Model Accuracy (%) Evaluated on CIFAR10 Under Connection Loss.							
Connection Error	User Capacity	Evaluated Model	*FedAvg*	*FedHetero*	*FjORD*	*FedSlim*	***FedResCuE***
$0.1 \le er \le 0.2$	uniform	$\boldsymbol{w}_{\times 1}$	50.36±2.17	-	61.79±1.62	57.31±1.27	**70.02±0.40**
		$\boldsymbol{w}_{\times 0.25}$	12.58±0.51	-	60.20±1.67	55.33±0.89	**67.40±0.84**
	cluster	$\boldsymbol{w}_{\times 1}$	-	60.92±1.33	64.52±0.60	62.35±1.76	**69.78±0.74**
		$\boldsymbol{w}_{\times 0.25}$	-	59.70±0.64	64.11±0.41	61.77±1.62	**68.83±1.00**
$0.2 \le er \le 0.3$	uniform	$\boldsymbol{w}_{\times 1}$	41.79±4.33	-	54.91±1.07	48.46±0.73	**64.80±0.85**
		$\boldsymbol{w}_{\times 0.25}$	12.27±0.93	-	55.20±1.31	48.42±0.87	**64.10±0.26**
	cluster	$\boldsymbol{w}_{\times 1}$	-	51.70±1.14	57.60±2.25	56.28±1.61	**65.74±0.30**
		$\boldsymbol{w}_{\times 0.25}$	-	51.90±0.33	57.69±2.22	56.34±1.56	**65.18±0.57**

Table 12: Performance under faulty connections, given 100% of training data.

Constant Error Rates er for DIGITSFIVE					
Domain	SVHN	Synthetic	USPS	MNIST	MNIST-M
er	0.05	0.05	0.3	0.3	0.05

DIGITSFIVE performance, trained using 5% of data.					
Domain	SVHN	Synthetic	USPS	MNIST	MNIST-M
	$\times$1.0 Model.				
Local	45.88±0.92	62.04±1.16	89.95±1.93	85.84±3.35	61.99±1.79
FedAvg	**69.54±0.15**	80.17±0.24	94.38±0.55	93.61±0.38	75.22±0.87
FedSlim	66.77±0.61	78.95±0.82	94.00±0.41	93.80±0.53	73.95±1.02
FjORD	49.72±23.42	57.12±30.11	68.60±36.34	68.01±37.35	56.29±26.48
FedResCuE	69.32±0.78	**80.57±0.77**	**95.17±0.47**	**95.05±0.44**	**76.89±0.76**
	$\times$0.25 Model.				
FedAvg	20.00±0.55	19.14±5.64	34.03±14.65	32.16±18.65	20.47±9.28
FedSlim	64.00±0.96	77.15±2.13	93.44±1.26	93.38±0.75	72.16±1.51
FjORD	48.11±22.35	54.58±31.15	68.51±36.28	65.79±39.13	52.75±29.16
FedResCuE	**67.11±0.77**	**79.06±0.41**	**94.55±0.37**	**94.64±0.28**	**75.64±0.37**

Table 13: *FedResCuE* is the most robust algorithm given heterogeneous data and domain-dependent connection errors.

A.4. Modeling of Connection Uncertainty

In our experiments, we assume that an error rate er is related to each model column transmission between the server and the edge device. This setting is built upon the mechanism of downstream wireless connections. Specifically, we present a fine-grained formulation of the connection error rates in a wireless communication scenario:

Lossy Wireless Connections: Following the wireless model of (Raza et al., 2020), we can leverage LTE-A (Kanchi et al., 2013), which is a representative model of 4G network, for the wireless links between the edge server (ES) and edge devices for FL, considering the orthogonal frequency division multiple access (OFDMA) scheme. The parameter $d_{es,k}$ denotes the distance between the edge server and the k^{th} edge device while the path loss between them can be characterized by $d_{es,k}^{-\sigma}$ and the white Gaussian noise power N_0, where σ is the path loss exponent. The wireless channel is modeled as a frequency-flat block-fading Rayleigh fading channel, with the uplink channel fading coefficient h (Wang et al., 2018). The uplink data rate of the k^{th} edge device is defined as:

$$R_k = B_k log_2 \left(1 + \frac{P_t d_{es,k}^{-\sigma} \left|h^2\right|}{N_0 + I}\right). \tag{6}$$

In the equation above, B_k denotes the channel bandwidth, P_t represents the transmission power of the ES. I is the inter-cell interference. As R_k fluctuates based on changing wireless network conditions, we can define a minimum uplink rate R_{min}, such that any rate lower than R_{min} will lead to timeouts and packet loss. As a result, **the probability that a packet gets lost over the edge network** shall be derived as the following:

$$\begin{aligned} Pr\left\{R_k < R_{min}\right\} &= Pr\left\{B_k log_2\left(1 + \frac{P_t d_{es,k}^{-\sigma}\left|h^2\right|}{N_0 + I}\right) < R_{min}\right\} \\ &= Pr\left\{d^{\sigma} > \frac{P_t\left|h^2\right|}{(N_0 + I)\,2^{\frac{R_{min}}{B_k}} - N_0 - I}\right\} \\ &= Pr\left\{d > \left(\frac{P_t\left|h^2\right|}{(N_0 + I)\,2^{\frac{R_{min}}{B_k}} - N_0 - I}\right)^{\frac{1}{\sigma}}\right\}. \end{aligned} \tag{7}$$

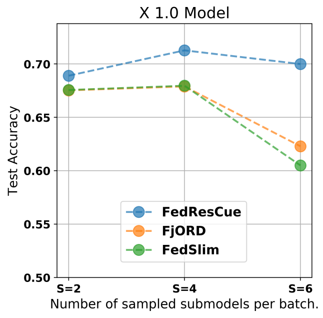
Suppose in a given LTE-A network, B_k, N_0, I, P_t, σ, and h are constants. If the distances between the edge devices and the ES, *i.e.*, $d_{es,k}$, follow the *Poisson* distribution, then the probability of a packet to be lost during transmission can be derived by Equation (7).

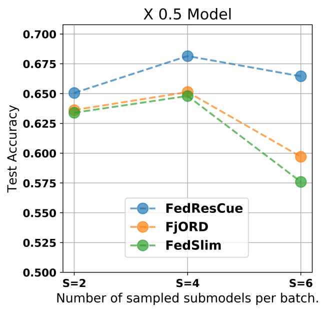
Macro-Level Simulation of Connection Uncertainty: When conducting experiments for unstable network connection scenarios, we use er, *i.e.* the connection loss rate for each *column*, as a macro-level modeling to approximate $Pr\left\{R_k < R_{min}\right\}$ in Equation (7). This type of modeling is grounded in that an upstream *column* and a downstream *packet* are logically related to each other. Depending on the size limit of a transmission packet and the granularity of our model decomposition, a column could be further decomposed into one or multiple packets during wireless transmission.

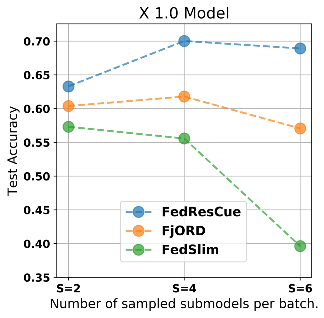
A.4.1. Ablation Study

Impacts of Sampling Frequency: We explored the effects of different sampling frequency S on model learning, where S is the number of submodels sampled per batch update (*e.g.*$|\hat{P}| = S$ in Algorithm 1). $S = 0$ would reduce all algorithms to regular *FedAvg* without self distillation. An overlarge S, on the other hand, may bring extra computation workload to edge devices. In our experiments, the sampling granularity is set to be $\times 0.05$ of the largest model width, and the smallest model width is set to be $\times 0.1$. Hence there are 19 eligible pruning ratios, *i.e.* $|\mathcal{P}| = 19$.

As shown in Figure 10, *FedResCuE* can benefit from finer-grained submodel sampling and maintains a robust performance across different choices of S. On the contrary, the performance of *FjORD* is only slightly improved by increasing S from 2 to 4, whose performance drops notably when S becomes overlarge, especially given insufficient training data. *FedSlim* is the most sensitive to the choice of S, which learns prunable submodels purely by knowledge distillation and batch-gradient updates. Its performance degrades drastically with an increasing S and becomes highly unstable under connection interruptions. We ascribe the robustness of *FedResCuE* over others to its progressive learning scheme, rather than a batch-gradient update strategy as *FedSlim* and *FjORD* applies.


(a) 20% training data, $\times 1.0$ model.


(b) 20% training data, $\times 0.5$ model.


(c) $0.1 \leq er \leq 0.2$, $\times 1.0$ model.

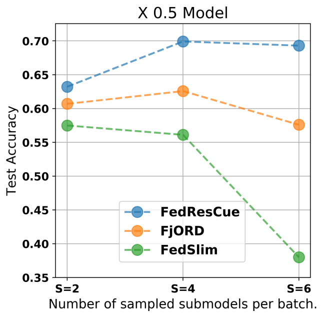

(d) $0.1 \leq er \leq 0.2$, $\times 0.5$ model.

Figure 10: Impacts of the submodel sampling frequency.

Effects of Knowledge Distillation: We compare *FedResCuE* against its ablated variant named *FedSeq* which does not require minimizing the KL-divergence between the largest model and the sampled submodel. We provide the model learning process of *FedSeq* in Algorithm 5. Table 14 summarizes their asymptotic performance under different FL settings, which demonstrates that *FedResCuE* is more beneficial to smaller submodels. We present the corresponding evaluation curves in Figure 11.

Algorithm 5 Local Model Update for *FedSeq*

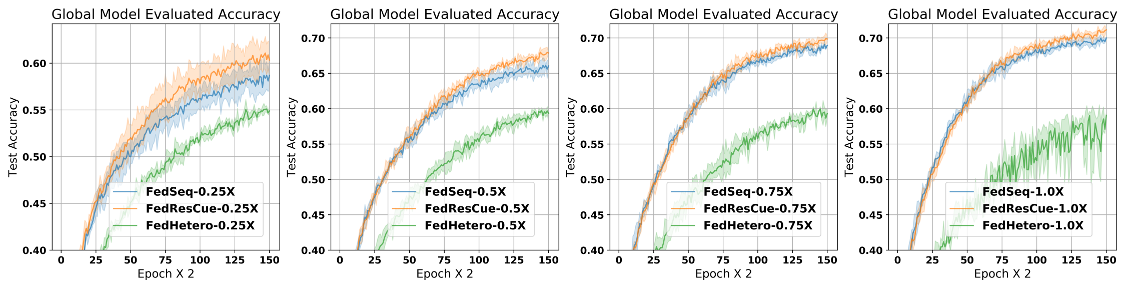
```
1: Inputs: Training dataset $D \subset \mathcal{X} \times \mathcal{Y}$; model with parameter set $\boldsymbol{\theta} \in \Theta$; pruning ratios $\mathcal{P}$, learning rate $\eta$, loss function $\mathcal{L}$, constant $S \leq |\mathcal{P}|$.
2: repeat
3:     Sample batch of $x, y \sim D$.
4:     Sample ordered ratios $\hat{P} = [p_i | p_i \in \mathcal{P}, p_i < p_{i+1} \, \forall i < S, p_S = 1]_{i=1}^{S}$
5:     for $p_i \sim \hat{P}$ do
6:         $g_i \leftarrow \nabla_{\{\boldsymbol{\theta}_{\times p_i} \setminus \boldsymbol{\theta}_{\times p_{i-1}}\}} \mathcal{L}(f(\mathcal{X}; \boldsymbol{\theta}_{\times p_i}), \mathcal{Y})$
7:         $\boldsymbol{\theta}_{\times p_i} \leftarrow \boldsymbol{\theta}_{\times p_i} - \eta * g_i$.
8:     end for
9:     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta * \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(x; \boldsymbol{\theta}), y)$.
10: until training stop
11: Return $\boldsymbol{\theta}$
```

Comparing *FedResCuE* and *FedSeq*, on CIFAR10 dataset

Algorithm	$\boldsymbol{w}_{\times 0.25}$	$\boldsymbol{w}_{\times 0.5}$	$\boldsymbol{w}_{\times 0.75}$	$\boldsymbol{w}_{\times 1.0}$
	Given 20 % training data			
FedResCuE	61.12±1.35	68.15±0.39	69.98±0.41	71.27±0.27
FedSeq	58.96±1.24	66.39±0.60	69.18±0.28	70.13±0.16
	$0.1 \leq er \leq 0.2$			
FedResCuE	67.40±0.84.	69.91±0.42.	70.23±0.42.	70.02±0.40
FedSeq	66.47±0.36	69.45±0.41	69.70±0.37	69.67±0.32

Table 14: Performance with and without knowledge-distillation.

Effects of Progressive Learning: One contributing factor to *FedResCuE*'s superior performance is its progressive learning strategy, which adaptively learns gradients by fixing the parameters of previously learned submodels. To validate the efficacy of progressive learning, we compared *FedResCuE* against a variant called *FedRush*The detailed model learning process of *FedRush* is provided in Algorithm 6. Learning curves of these two algorithms are illustrated in Figure 12, which demonstrate that the progressive parameter update, as *FedResCuE* adopts, is necessary to derive reliable models, especially given insufficient training data or connection interruptions. *FedRush*, on the other hand, undermines the knowledge learned by smaller submodels by overwriting parameters during the model update, which could otherwise serve as a good initialization for the subsequent submodel to build up representative domain knowledge.

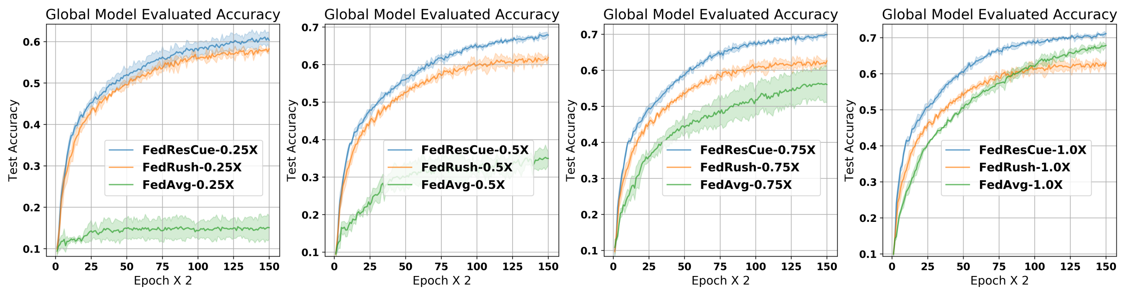


(a) $\times_{0.25}$ model. (b) $\times_{0.5}$ model. (c) $\times_{0.75}$ model. (d) $\times_{1.0}$ model.

Figure 11: Compared with *FedSeq*, the knowledge-distillation strategy in *FedResCuE* can benefit smaller submodels.

Algorithm 6 Local Model Update for *FedRush*

```
 1: Inputs: training dataset $D \subset \mathcal{X} \times \mathcal{Y}$; model with parameter set $\boldsymbol{\theta}$; pruning ratios $\mathcal{P}$, learning rate $\eta$, loss function $\mathcal{L}$,
    constant $S \leq |\mathcal{P}|$.
 2: repeat
 3:     Sample batch $x, y \sim D$.
 4:     Sample ordered ratios $\hat{P} = [p_i | p_i \in \mathcal{P}, p_i < p_{i+1} \; \forall i < S, p_S = 1]_{i=1}^{S}$
 5:     $\bar{\boldsymbol{\theta}} \leftarrow$ stop_gradient$(\boldsymbol{\theta})$, $\boldsymbol{\theta}_{\times 0} = \emptyset$.
 6:     for $p_i \sim \hat{P}$ do
 7:         $g_i \leftarrow \nabla_{\boldsymbol{\theta}_{\times p_i}} \left\{ \mathcal{L}(f(\mathcal{X}; \boldsymbol{\theta}_{\times p_i}), \mathcal{Y}) + D_{\mathrm{KL}}[f(x; \bar{\boldsymbol{\theta}}) \| f(x; \boldsymbol{\theta}_{\times p_i})] \right\}$ .
 8:         $\boldsymbol{\theta}_{\times p_i} \leftarrow \boldsymbol{\theta}_{\times p_i} - \eta * g_i$.    ▷ (Overwrite previously learned $\boldsymbol{\theta}_{p_{i-1}}$.)
 9:     end for
10:     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta * \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(x; \boldsymbol{\theta}), y)$
11: until training stop
```



(a) $\times_{0.25}$ model. (b) $\times_{0.5}$ model. (c) $\times_{0.75}$ model. (d) $\times_{1.0}$ model.

Figure 12: Compared to *FedRush*, *FedResCuE* maintains a high performance for the $\times 1.0$ model.

A.5. Overview of Model Learning Performance

Performance with Stable Network Connections:

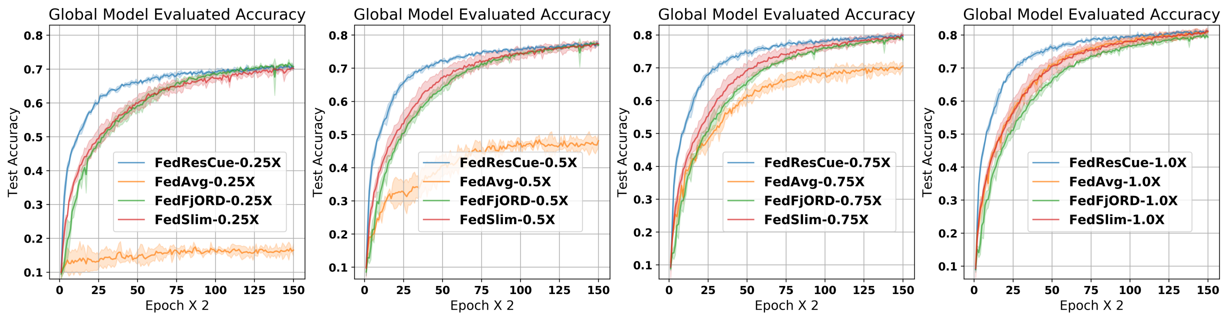
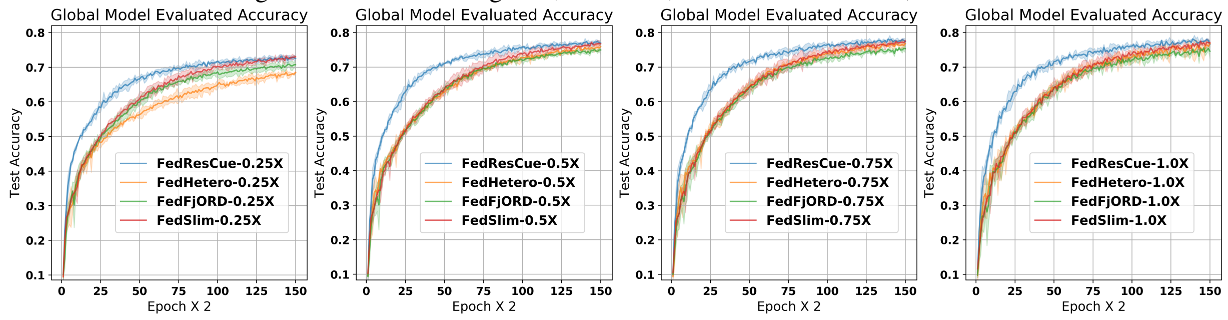


Figure 13: 100% training data, CIFAR10, **uniform** architecture, $er = 0$.



(a) $\times 0.25$ model (b) $\times 0.5$ model (c) $\times 0.75$ model (d) $\times 1.0$ model

Figure 14: 100% training data, CIFAR10, **cluster** architecture, $er = 0$.

Performance Given Insufficient Training Data:

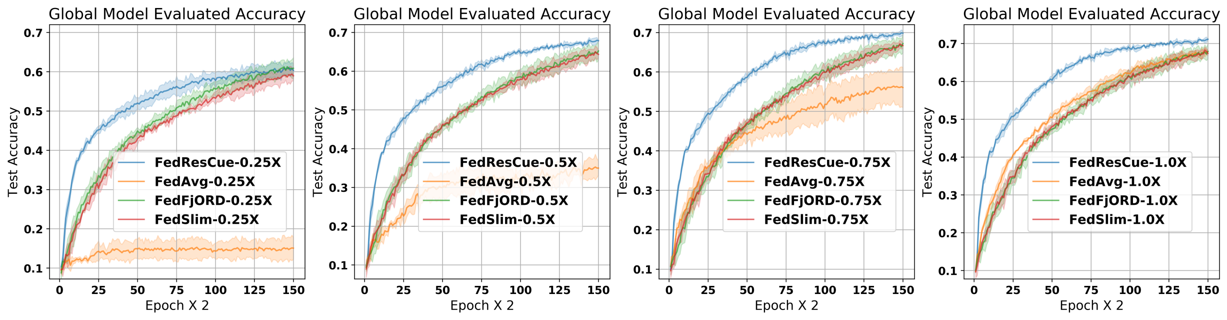
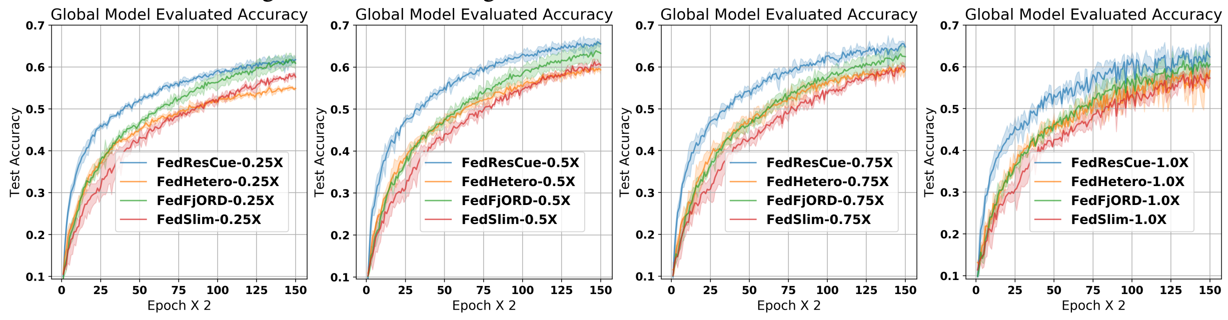


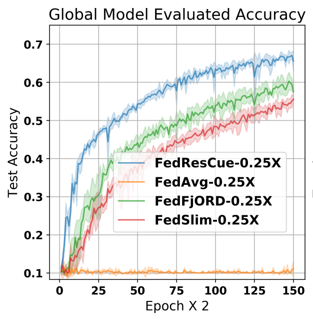
Figure 15: 20% training data, CIFAR10, **uniform** architecture, $er = 0$.



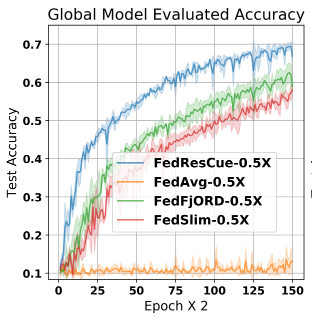
(a) $\times 0.25$ model (b) $\times 0.5$ model (c) $\times 0.75$ model (d) $\times 1.0$ model

Figure 16: 20% training data, CIFAR10, **cluster** architecture, $er = 0$.

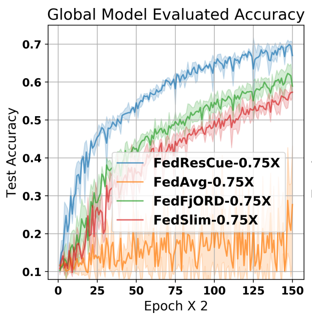
Performance Under Connection Loss:



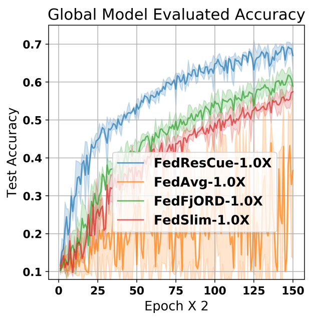
(a) $\times 0.25$ model



(b) $\times 0.5$ model

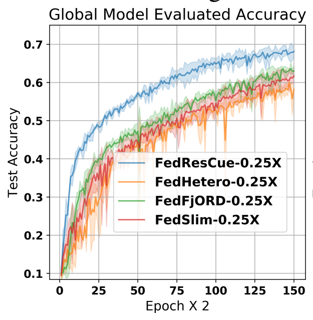


(c) $\times 0.75$ model

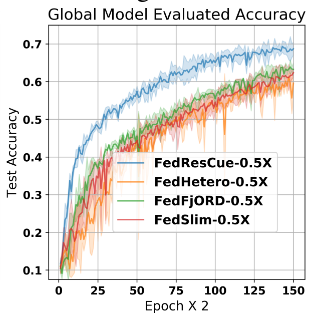


(d) $\times 1.0$ model

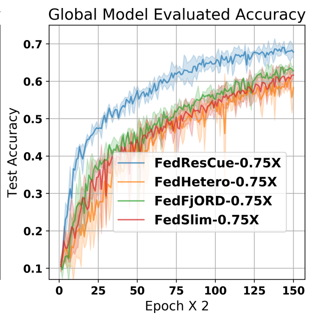
Figure 17: 100% training data, CIFAR10, **uniform** architecture, $0.1 \leq er \leq 0.2$.



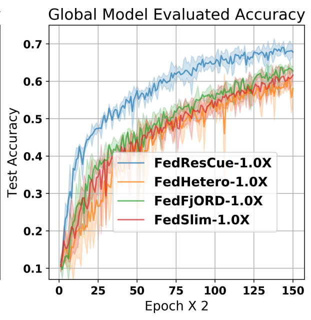
(a) $\times 0.25$ model



(b) $\times 0.5$ model

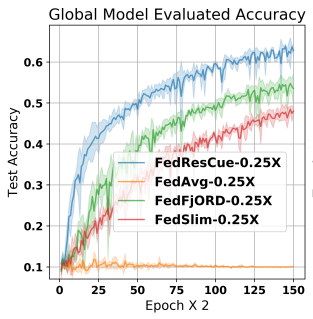


(c) $\times 0.75$ model

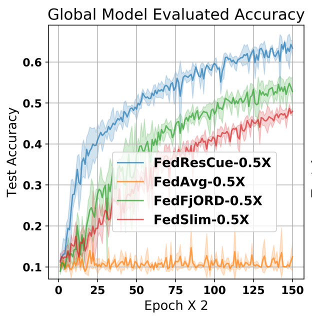


(d) $\times 1.0$ model

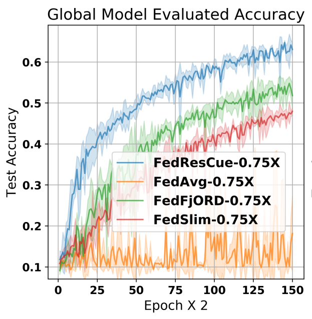
Figure 18: 100% training data, CIFAR10, **cluster** architecture, $0.1 \leq er \leq 0.2$.



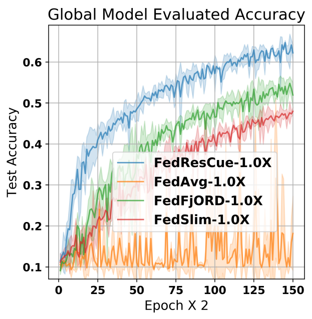
(a) $\times 0.25$ model



(b) $\times 0.5$ model

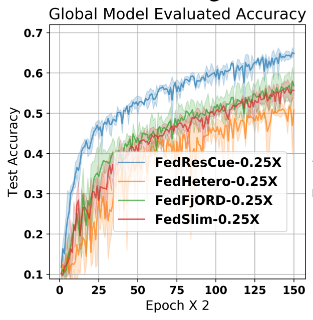


(c) $\times 0.75$ model

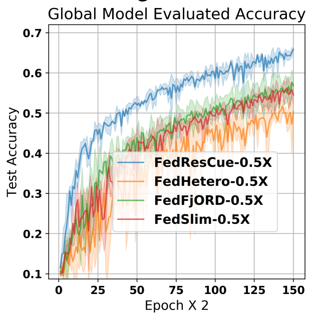


(d) $\times 1.0$ model

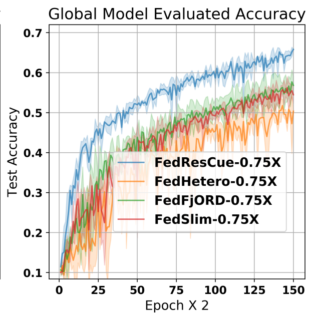
Figure 19: 100% training data, CIFAR10, **uniform** architecture, $0.2 \leq er \leq 0.3$.



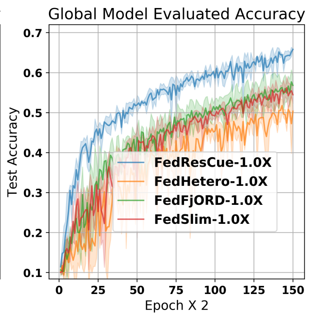
(a) $\times 0.25$ model



(b) $\times 0.5$ model



(c) $\times 0.75$ model



(d) $\times 1.0$ model

Figure 20: 100% training data, CIFAR10, **cluster** architecture, $0.2 \leq er \leq 0.3$.

A.6. Experiment Configurations

A.6.1. Model Architecture

For the CIFAR10 domain, we build a ResNet neural network (He et al., 2016) using 4 *residual blocks*, while a residual block maintains 1) a convolution module that consits of 3 CONV2D layers, each followed by a BATCHNORM2D layer and a RELU activation layer; and 2) a *shortpath* module to be in parallel with the convolution module. The ResNet model contains 8,036,426 trainable parameters in total, with a model size of 33.09 MB. For the DIGITSFIVE domains, we build a neural network consiting of 3 CONV2D layers, each followed by a BATCHNORM layer, and 3 LINEAR layers. It contains 14,214,090 trainable parameters in total, with a model size of 55.30 MB.

Network Architecture for Learning **CIFAR10**.		
Layer	Output Shape	# of Parameters
Conv2d-1	64 × 14 × 14	9,408
MaxPool2d-4	64 × 7 × 7	0
Conv2d-5	64 × 7 × 7	4,096
Conv2d-8	64 × 7 × 7	36,864
Conv2d-11	256 × 7 × 7	16,384
Conv2d-13	256 × 7 × 7	16,384
Conv2d-17	128 × 7 × 7	32,768
Conv2d-20	128 × 4 × 4	147,456
Conv2d-23	512 × 4 × 4	65,536
Conv2d-25	512 × 4 × 4	131, 072
Conv2d-29	256 ×4 × 4	131,072
Conv2d-32	256 × 2 × 2	589,824
Conv2d-35	1024 × 2 × 2	262,144
Conv2d-37	1024 × 2 × 2	524,288
Conv2d-41	512 × 2, 2	524,288
Conv2d-44	512 × 1 × 1	2,359,296
Conv2d-47	2048 × 1 × 1	1,048,576
Conv2d-49	2048 × 1 × 1	2,097,152
AvgPool2d-53	2048 × 1 × 1	0
Linear-54	10	20,490

Table 15: ResNet Architecture for Learning CIFAR10, omitting BatchNorm and ReLU layers.

Network Architecture for Learning **DIGITSFIVE**.		
Layer	Output Shape	# of Parameters
Conv2d-1	64 × 28 × 28	4,864
BatchNorm2d-2	64 × 28 × 28	128
Conv2d-3	64 × 14 × 14	102,464
BatchNorm2d-4	64 × 14 × 14	128
Conv2d-5	128 × 7 × 7	204,928
BatchNorm2d-6	128 × 7 × 7	256
Linear-7	2048	12,847,104
Linear-8	512	1,049,088
Linear-9	10	5,130

Table 16: Model Architecture for Learning DIGITSFIVE.

A.6.2. Optimizer Implementation for Progressive Learning

In practice, we customzie the default SGD optimizer implemented in Pytorch to realize progressive gradient updates. A trainable neural layer consits of parameter tensors, each of which can be treated as a weight matrix. When calculating gradients for the neural layer, we specify the *columns* to be updated, which corresponds to a set of indices for elements in the weight matrix. When applying the gradients, we *mask* out the gradients of parameters that were not included in the specified columns.

A.6.3. Hyper-parameter Configurations

We summarize in Table 17 the hyper-parameters used in our experiments.

Hyper-parameter Configurations		
Domain	Hyper-parameter	Value
Shared	Optimizer	SGD
	learning rate	0.1
	Momentum	0.9
	Nesterov	TRUE
	Weight decay	10^{-4}
	Track training in BatchNorm	FALSE
	Share BatchNorm	TRUE
	Data category	10
	# of active users	5
	Random seeds for training	3, 5, 7
	Batch Size	32
CIFAR10	Training Epoch	300
	# of total users	20
	Used training data	100%, 20%
	Column Granularity for $\mathcal{P}$	$\times 0.05$
DIGITSFIVE	Training Epoch	100
	# of total users	10
	# users per domain	2
	Used training data	5%
	Column Granularity for $\mathcal{P}$	$\times 0.125$

Table 17: Configurations of Hyper-parameters.