FjORD: Fair and Accurate Federated Learning under heterogeneous targets with Ordered Dropout

Samuel Horváth[*]
KAUST[†]
Thuwal, KSA
samuel.horvath@kaust.edu.sa

Stefanos Laskaridis[*]
Samsung AI Center
Cambridge, UK
mail@stefanos.cc

Mario Almeida[*]
Samsung AI Center
Cambridge, UK
mario.a@samsung.com

Ilias Leontiadis
Samsung AI Center
Cambridge, UK
i.leontiadis@samsung.com

Stylianos I. Venieris
Samsung AI Center
Cambridge, UK
s.venieris@samsung.com

Nicholas D. Lane
Samsung AI Center &
University of Cambridge
Cambridge, UK
nic.lane@samsung.com

Abstract

Federated Learning (FL) has been gaining significant traction across different ML tasks, ranging from vision to keyboard predictions. In large-scale deployments, client heterogeneity is a fact and constitutes a primary problem for fairness, training performance and accuracy. Although significant efforts have been made into tackling statistical data heterogeneity, the diversity in the processing capabilities and network bandwidth of clients, termed as system heterogeneity, has remained largely unexplored. Current solutions either disregard a large portion of available devices or set a uniform limit on the model's capacity, restricted by the least capable participants. In this work, we introduce Ordered Dropout, a mechanism that achieves an ordered, nested representation of knowledge in deep neural networks (DNNs) and enables the extraction of lower footprint submodels without the need of retraining. We further show that for linear maps our Ordered Dropout is equivalent to SVD. We employ this technique, along with a self-distillation methodology, in the realm of FL in a framework called FjORD. FjORD alleviates the problem of client system heterogeneity by tailoring the model width to the client's capabilities. Extensive evaluation on both CNNs and RNNs across diverse modalities shows that FjORD consistently leads to significant performance gains over state-of-the-art baselines, while maintaining its nested structure.

1 Introduction

Over the past few years, advances in deep learning have revolutionised the way we interact with everyday devices. Much of this success relies on the availability of large-scale training infrastructures and the collection of vast amounts of training data. However, users and providers are becoming increasingly aware of the privacy implications of this ever-increasing data collection, leading to the creation of various privacy-preserving initiatives by service providers [3] and government regulators [13].

Federated Learning (FL) [49] is a relatively new subfield of machine learning (ML) that allows the training of models without the data leaving the users' devices; instead, FL allows users to collaboratively train a model by moving the computation to them. At each round, participating devices download the latest model and compute an updated model using their local data. These locally trained models are then sent from the participating devices back to a central server where updates are

[*]Indicates equal contribution.
[†]Work while intern at Samsung AI Center.

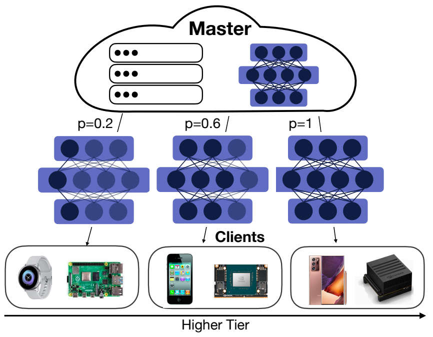


Figure 1: FjORD employs OD to tailor the amount of computation to the capabilities of each participating device.

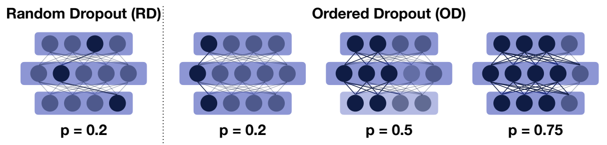


Figure 2: Ordered vs. Random Dropout. In this example, the left-most features are used by more devices during training, creating a natural ordering to the importance of these features.

aggregated for next round's global model. Until now, a lot of research effort has been invested with the sole goal of maximising the accuracy of the global model [49, 45, 42, 34, 66], while complementary mechanisms have been proposed to ensure privacy and robustness [7, 17, 50, 51, 30, 5].

A key challenge of deploying FL in the wild is the vast *heterogeneity of devices* [41], ranging from low-end IoT to flagship mobile devices. Despite this fact, the widely accepted norm in FL is that the local models have to share the *same* architecture as the global model. Under this assumption, developers typically opt to either drop low-tier devices from training, hence introducing training bias due to unseen data [33], or limit the global model's size to accommodate the slowest clients, leading to degraded accuracy due to the restricted model capacity [10]. In addition to these limitations, variability in sample sizes, computation load and data transmission speeds further contribute to a very unbalanced training environment. Finally, the resulting model might not be as efficient as models specifically tailored to the capabilities of each device tier to meet the minimum processing-performance requirements [37].

In this work, we introduce FjORD (Fig. 1), a novel adaptive training framework that enables heterogeneous devices to participate in FL by dynamically adapting model size – and thus computation, memory and data exchange sizes – to the available client resources. To this end, we introduce Ordered Dropout (OD), a mechanism for run-time *ordered (importance-based) pruning*, which enables us to extract and train submodels in a nested manner. As such, OD enables *all* devices to participate in the FL process independently of their capabilities by training a submodel of the original DNN, while still contributing knowledge to the global model. Alongside OD, we propose a *self-distillation* method from the maximal supported submodel on a device to enhance the feature extraction of smaller submodels. Finally, our framework has the additional benefit of producing models that can be dynamically scaled during inference, based on the hardware and load constraints of the device.

Our evaluation shows that FjORD enables significant accuracy benefits over the baselines across diverse datasets and networks, while allowing for the extraction of *submodels* of varying FLOPs and sizes *without* the need for *retraining*.

2 Motivation

Despite the progress on the accuracy front, the unique deployment challenges of FL still set a limit to the attainable performance. FL is typically deployed on either siloed setups, such as among hospitals, or on mobile devices in the wild [8]. In this work, we focus on the latter setting. Hence, while cloud-based distributed training uses powerful high-end clients [22], in FL these are commonly substituted by resource-constrained and heterogeneous embedded devices.

In this respect, FL deployment is currently hindered by the vast *heterogeneity* of client hardware [69, 31, 8]. On the one hand, different mobile hardware leads to significantly varying processing speed [1], in turn leading to longer waits upon aggregation of updates (*i.e.* stragglers). At the same time, devices of mid and low tiers might not even be able to support larger models, *e.g.* the model does not fit in memory or processing is slow, and, thus, are either excluded or dropped upon timeouts from the training process, together with their unique data. More interestingly, the resource allocation to participating devices may also reflect on demographic and socio-economic information of owners, that makes the exclusion of such clients unfair [33] in terms of participation. Analogous to the device load and heterogeneity, a similar trend can be traced in the downstream (model) and upstream (updates) network communication in FL, which can be an additional substantial bottleneck for the training procedure [58].

3 Ordered Dropout

In this paper, we firstly introduce the tools that act as enablers for heterogeneous federated training. Concretely, we have devised a mechanism of importance-based pruning for the easy extraction of subnetworks from the original, specially trained model, each with a different computational and memory footprint. We name this technique **Ordered Dropout** (OD), as it orders knowledge representation in *nested submodels* of the original network.

More specifically, our technique starts by sampling a value (denoted by p) from a distribution of candidate values. Each of these values corresponds to a specific submodel, which in turn gets translated to a specific computational and memory footprint (see Table 1b). Such sampled values and associations are depicted in Fig. 2. Contrary to conventional dropout (RD), our technique drops adjacent components of the model instead of random neurons, which translates to computational benefits[3] in today's linear algebra libraries and higher accuracy as shown later.

3.1 Ordered Dropout Mechanics

The proposed OD method is parametrised with respect to: i) the value of the dropout rate $p \in (0, 1]$ per layer, ii) the set of candidate values $\mathcal{P}$, such that $p \in \mathcal{P}$ and iii) the sampling method of p over the set of candidate values, such that $p \sim D_{\mathcal{P}}$, where $D_{\mathcal{P}}$ is the distribution over $\mathcal{P}$.

A primary hyperparameter of OD is the dropout rate p which defines how much of each layer is to be included, with the rest of the units dropped in a structured and ordered manner. The value of p is selected by sampling from the dropout distribution $D_{\mathcal{P}}$ which is represented by a set of discrete values $\mathcal{P} = \{s_1, s_2, \ldots, s_{|\mathcal{P}|}\}$ such that $0<s_1<\ldots<s_{|\mathcal{P}|} \leq 1$ and probabilities $\mathbf{P}(p = s_i) > 0,\ \forall i \in [|\mathcal{P}|]$ such that $\sum_{i=1}^{|\mathcal{P}|} \mathbf{P}(p = s_i) = 1$. For instance, a uniform distribution over $\mathcal{P}$ is denoted by $p \sim \mathcal{U}_{\mathcal{P}}$ (*i.e.* $D = \mathcal{U}$). In our experiments we use uniform distribution over the set $\mathcal{P} = \{i/k\}_{i=1}^{k}$, which we refer to as $\mathcal{U}_k$ (or *uniform-k*). The discrete nature of the distribution stems from the innately discrete number of neurons or filters to be selected. The selection of set $\mathcal{P}$ is discussed in the next subsection.

The dropout rate p can be constant across all layers or configured individually per layer l, leading to $p_l \sim D_{\mathcal{P}}^{l}$. As such an approach opens the search space dramatically, we refer the reader to NAS techniques [72] and continue with the same p value across network layers for simplicity, without hurting the generality of our approach.

Given a p value, a pruned p-subnetwork can be directly obtained as follows. For each[4] layer l with width[5] K_l , the submodel for a given p has all neurons/filters with index $\{0, 1, \ldots, \lceil p \cdot K_l \rceil - 1\}$ included and $\{\lceil p \cdot K_l \rceil, \ldots, K_l - 1\}$ pruned. Moreover, the unnecessary connections between pruned neurons/filters are also removed[6]. We denote a pruned p-subnetwork $\mathbf{F}_p$ with its weights $\boldsymbol{w}_p$, where $\mathbf{F}$ and $\boldsymbol{w}$ are the original network and weights, respectively. Importantly, contrary to existing pruning techniques [21, 38, 52], a p-subnetwork from OD can be directly obtained post-training without the need to fine-tune, thus eliminating the requirement to access any labelled data.

3.2 Training OD Formulation

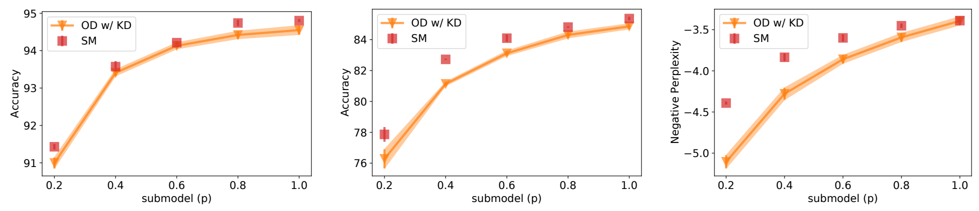
We propose two ways to train an OD-enabled network: i) *plain OD* and ii) *knowledge distillation OD* training (OD w/ KD). In the first approach, in each step we first sample $p \sim D_{\mathcal{P}}$; then we perform the forward and backward pass using the p-reduced network $\mathbf{F}_p$; finally we update the submodel's weights using the selected optimiser. Since sampling a p-reduced network provides us significant computational savings on average, we can exploit this reduction to further boost accuracy. Therefore, in the second approach we exploit the nested structure of OD, *i.e.* $p_1 < p_2 \implies \mathbf{F}_{p_1} \subset \mathbf{F}_{p_2}$ and allow for the bigger capacity supermodel to teach the sampled p-reduced network at each iteration

[3]OD, through its nested pruning scheme that requires neither additional data structures for bookkeeping nor complex and costly data layout transformations, can capitalise directly over the existing and highly optimised dense matrix multiplication libraries.

[4]Note that p affects the number of output channels/neurons and thus the number of input channels/neurons of the next layer. Furthermore, OD is not applied on the input and last layer to maintain the same dimensionality.

[5]*i.e.* neurons for fully-connected layers (linear and recurrent) and filters for convolutional layers. RNN cells can be seen as a set of linear feedforward layers with activation and composition functions.

[6]For BatchNorm, we maintain a separate set of statistics for every dropout rate p. This has only a marginal effect on the number of parameters and can be used in a privacy-preserving manner [44].



(a) ResNet18 - CIFAR10 (b) CNN - EMNIST (c) RNN - Shakespeare

Figure 3: Full non-federated datasets. OD-Ordered Dropout with $D_{\mathcal{P}} = \mathcal{U}_5$, SM-single independent models, KD-knowledge distillation.

via knowledge distillation (teacher $p_{\max} > p$, $p_{\max} = \max \mathcal{P}$). In particular, in each iteration, the loss function consists of two components as follows:

$$\mathcal{L}_d(\text{SM}_p, \text{SM}_{p_{\max}}, \boldsymbol{y}_{\text{label}}) = (1-\alpha)\text{CE}(\max(\text{SM}_p), \boldsymbol{y}_{\text{label}}) + \alpha\text{KL}(\text{SM}_p, \text{SM}_{p_{\max}}, T)$$

where SM_p is the *softmax* output of the sampled p-submodel, $\boldsymbol{y}_{\text{label}}$ is the ground-truth label, CE is the cross-entropy function, KL is the KL divergence, T is the distillation temperature [24] and α is the relative weight of the two components. We observed in our experiments always backpropagating also the teacher network further boosts performance. Furthermore, the best performing values for distillation were $\alpha = T = 1$, thus smaller models exactly mimic the teacher output. This means that new knowledge propagates in submodels by proxy, *i.e.* by backpropagating on the teacher, leading to the following loss function:

$$\mathcal{L}_d(\text{SM}_p, \text{SM}_{p_{\max}}, \boldsymbol{y}_{\text{label}}) = \text{KL}(\text{SM}_p, \text{SM}_{p_{\max}}, T) + \text{CE}(\max(\text{SM}_{p_{\max}}), \boldsymbol{y}_{\text{label}})$$

3.3 Ordered Dropout exactly recovers SVD

We further show that our new OD formulation can recover the Singular Value Decomposition (SVD) in the case where there exists a linear mapping from features to responses. We formalise this claim in the following theorem.

Theorem 1. *Let $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^m$ be a NN with two fully-connected linear layers with no activation or biases and $K = \min\{m, n\}$ hidden neurons. Moreover, let data $\mathcal{X}$ come from a uniform distribution on the n-dimensional unit ball and A be an $m \times n$ full rank matrix with K distinct singular values. If response y is linked to data $\mathcal{X}$ via a linear map: $x \to Ax$ and distribution $D_{\mathcal{P}}$ is such that for every $b \in [K]$ there exists $p \in \mathcal{P}$ for which $b = \lceil p \cdot K \rceil$, then for the optimal solution of*

$$\min_{\mathbf{F}} \mathbb{E}_{x\sim\mathcal{X}} \mathbb{E}_{p\sim D_{\mathcal{P}}} \|\mathbf{F}_p(x) - y\|^2$$

it holds $\mathbf{F}_p(x) = A_b x$, where A_b is the best b-rank approximation of A and $b = \lceil p \cdot K \rceil$.

Theorem 1 shows that our OD formulation exhibits not only intuitively, but also theoretically ordered importance representation. Proof of this claim is deferred to the Appendix.

3.4 Model-Device Association

Computational and Memory Implications. The primary objective of OD is to alleviate the excessive *computational* and *memory* demands of the training and inference deployments. When a layer is shrunk through OD, there is no need to perform the forward and backward passes or gradient updates on the pruned units. As a result, OD offers gains both in terms of FLOP count and model size. In particular, for every fully-connected and convolutional layer, the number of FLOPs and weight parameters is reduced by $K_1 \cdot K_2 / \lceil p \cdot K_1 \rceil \cdot \lceil p \cdot K_2 \rceil \sim 1/p^2$, where K_1 and K_2 correspond to the number of input and output neurons/channels, respectively. Accordingly, the bias terms are reduced by a factor of $K_2 / \lceil p \cdot K_2 \rceil \sim 1/p$. The normalisation, activation and pooling layers are compressed in terms of FLOPs and parameters similarly to the biases in fully-connected and convolutional layers. This is also evident in Table 1b. Finally, smaller model size also leads to reduced memory footprint for gradients and the optimiser's state vectors such as momentum. However, how are these submodels related to devices in the wild and how is this getting modelled?

Ordered Dropout Rates Space. Our primary objective with OD is to tackle *device heterogeneity*. Inherently, each device has certain capabilities and can run a specific number of model operations within a given time budget. Since each p value defines a submodel of a given width, we can indirectly associate a $p^i_{\max}$ value with the i-th device capabilities, such as memory, processing throughput or energy budget. As such, each participating client is given at most the $p^i_{\max}$-submodel it can handle.

Devices in the wild, however, can have dramatically different capabilities; a fact further exacerbated by the co-existence of previous-generation devices. Modelling discretely each device becomes quickly intractable at scale. Therefore, we cluster devices of similar capabilities together and subsequently associate a single p_{max}^i value with each cluster. This clustering can be done heuristically (*i.e.* based on the specifications of the device) or via benchmarking of the model on the actual device and is considered a system-design decision for our paper. As smartphones nowadays run a multitude of simultaneous tasks [46], our framework can further support modelling of transient device load by reducing its associated p_{max}^i, which essentially brings the capabilities of the device to a lower tier at run time, thus bringing real-time adaptability to FjORD.

Concretely, the discrete candidate values of $\mathcal{P}$ depend on i) the number of clusters and corresponding device tiers, ii) the different load levels being modelled and iii) the size of the network itself, as *i.e.* for each tier i there exists p_{max}^i beyond which the network cannot be resolved. In this paper, we treat the former two as invariants (assumed to be given by the service provider), but provide results across different number and distributions of clusters, models and datasets.

3.5 Preliminary Results

Here, we present some results to showcase the performance of OD in the *centralised* non-FL training setting (i.e. the server has access to all training data) across three tasks, explained in detail in § 5. Concretely, we run OD with distribution $D_{\mathcal{P}} = \mathcal{U}_5$ (uniform distribution over the set $\{i/5\}_{i=1}^5$) and compare it with end-to-end trained submodels (SM) trained in isolation for the given width of the model. Fig. 3 shows that across the three datasets, the best attained performance of OD along every width p is very close to the performance of the baseline models. We extend this comparison against Random Dropout in the Appendix. We note at this point that the submodel baselines are trained from scratch, explicitly optimised to that given width with no possibility to jump across them, while our OD model was trained using a single training loop and offers the ability to switch between accuracy-computation points without the need to retrain.

4 FjORD

Building upon the shoulders of OD, we introduce FjORD, a framework for federated training over *heterogenous* clients. We subsequently describe the FjORD's workflow, further documented in Alg. 1.

As a starting point, the global model architecture, $\mathbf{F}$, is initialised with weights $\boldsymbol{w}^0$, either randomly or via a pretrained network. The dropout rates space $\mathcal{P}$ is selected along with distribution $D_{\mathcal{P}}$ with $|\mathcal{P}|$ discrete candidate values, with each p corresponding to a subnetwork of the global model with varying FLOPs and parameters. Next, the devices to participate are clustered into $|\mathcal{C}_{\text{tiers}}|$ tiers and a p_{max}^c value is associated with each cluster c. The resulting p_{max}^c represents the maximum capacity of the network that devices in this cluster can handle without violating a latency or memory constraint.

At the beginning of each communication round t, the set of participating devices $\mathcal{S}_t$ is determined, which either consists of all available clients $\mathcal{A}_t$ or contains only a random subset of $\mathcal{A}_t$ based on the server's capacity. Next, the server broadcasts the current model to the set of clients $\mathcal{S}_t$ and each client i receives $\boldsymbol{w}_{p_{\text{max}}^i}$. On the client side, each client runs E local iterations and at each local iteration k, the device i samples $p_{(i,k)}$ from conditional distribution $D_{\mathcal{P}}|D_{\mathcal{P}} \leq p_{\text{max}}^i$ which accounts for its limited capability. Subsequently, each client updates the respective weights ($\boldsymbol{w}_{p_{(i,k)}}$) of the local submodel using the `FedAvg` [49] update rule. In this step, other strategies [42, 66, 34] can be interchangeably employed. At the end of the local iterations, each device sends its update back to the server.

Finally, the server aggregates these communicated changes and updates the global model, to be distributed in the next global federated round to a different subset of devices. Heterogeneity of devices leads to heterogeneity in the model updates and, hence, we need to account for that in the global aggregation step. To this end, we utilise the following aggregation rule

$$\boldsymbol{w}_{s_j}^{t+1} \setminus \boldsymbol{w}_{s_{j-1}}^{t+1} = \text{WA}\left(\left\{\boldsymbol{w}_{i_{s_j}}^{(i,t,E)} \setminus \boldsymbol{w}_{s_{j-1}}^{(i,t,E)}\right\}_{i\in\mathcal{S}_t^j}\right) \tag{1}$$

where $\boldsymbol{w}_{s_j} \setminus \boldsymbol{w}_{s_{j-1}}$ are the weights that belong to $\mathbf{F}_{s_j}$ but not to $\mathbf{F}_{s_{j-1}}$, $\boldsymbol{w}^{t+1}$ the global weights at communication round $t+1$, $\boldsymbol{w}^{(i,t,E)}$ the weights on client i at communication round t after E local iterations, $\mathcal{S}_t^j = \{i \in \mathcal{S}_t : p_{\text{max}}^i \geq s_j\}$ a set of clients that have the capacity to update $\boldsymbol{w}_{s_j}$, and WA stands for weighted average, where weights are proportional to the amount of data on each client.

Communication Savings. In addition to the computational savings (§3.4), OD provides additional *communication* savings. First, for the server-to-client transfer, every device with $p_{\text{max}}^i < 1$ observes a reduction of approximately $1/(p_{\text{max}}^i)^2$ in the downstream transferred data due to the smaller model size (§ 3.4). Accordingly, the upstream client-to-server transfer is decreased by $1/(p_{\text{max}}^i)^2$ as only the gradient updates of the unpruned units are transmitted.

Algorithm 1: FjORD (Proposed Framework)

```
Input: $\mathbf{F}, \boldsymbol{w}^0, D_{\mathcal{P}}, T, E$
for $t \leftarrow 0$ to $T-1$ do // Global rounds
    Server selects clients as a subset $\mathcal{S}_t \subset \mathcal{A}_t$
    Server broadcasts weights of $p^i_{\text{max}}$-submodel to each client $i \in \mathcal{S}_t$
    for $k \leftarrow 0$ to $E-1$ do // Local iterations
        $\forall i \in \mathcal{S}_t$: Device $i$ samples $p_{(i,k)} \sim D_{\mathcal{P}} | D_{\mathcal{P}} \leq p^i_{\text{max}}$ and updates the weights of local model
    end
    $\forall i \in \mathcal{S}_t$: device $i$ sends to the server the updated weights $\boldsymbol{w}^{(i,t,E)}$
    Server updates $\boldsymbol{w}^{t+1}$ as in Eq. (1)
end
```

Identifiability. A standard procedure in FL is to perform element-wise averaging to aggregate model updates from clients. However, coordinate-wise averaging of updates may have detrimental effects on the accuracy of the global model, due to the permutation invariance of the hidden layers. Recent techniques tackle this problem by matching clients' neurons before averaging [71, 60, 65]. Unfortunately, doing so is computationally expensive and hurts scalability. FjORD mitigates this issue since it exhibits the natural importance of neurons/channels within each hidden layer by design; essentially OD acts in lieu of a neuron matching algorithm without the computational overhead.

Subnetwork Knowledge Transfer. In § 3.2, we introduced knowledge distillation for our OD formulation. We extend this approach to FjORD, where instead of the full network, we employ width $\max\{p \in \mathcal{P} : p \leq p^i_{\text{max}}\}$ as a teacher network in each local iteration on device i. We provide the alternative of FjORD *with* knowledge distillation mainly as a solution for cases where the client bottleneck is memory- or network-related, rather than computational in nature [35]. However, in cases where client devices are computationally bound in terms of training latency, we propose FjORD *without* KD or decreasing p^i_{max} to account for the overhead of KD.

5 Evaluation of FjORD

In this section, we provide a thorough evaluation of FjORD and its components across different tasks, datasets, models and device cluster distributions to show its performance, elasticity and generality.

Datasets and Models. We evaluate FjORD on two vision and one text prediction task, shown in Table 1a. For CIFAR10 [36], we use the "CIFAR" version of ResNet18 [23]. We federate the dataset by randomly dividing it into equally-sized partitions, each allocated to a specific client, and thus remaining IID in nature. For FEMNIST, we use a CNN with two convolutional layers followed by a softmax layer. For Shakespeare, we employ a RNN with an embedding layer (without dropout) followed by two LSTM [25] layers and a softmax layer. We report the model's performance of the last epoch on the test set which is constructed by combining the test data for each client. We report top-1 accuracy vision tasks and negative perplexity for text prediction. Further details, such as hyperparameters, description of datasets and models are available in the Appendix.

Table 1: Datasets and models

Dataset	Model	# Clients	# Samples	Task
CIFAR10	ResNet18	100	$50,000$	Image classification
FEMNIST	CNN	$3,400$	$671,585$	Image classification
Shakespeare	RNN	715	$38,001$	Next character prediction

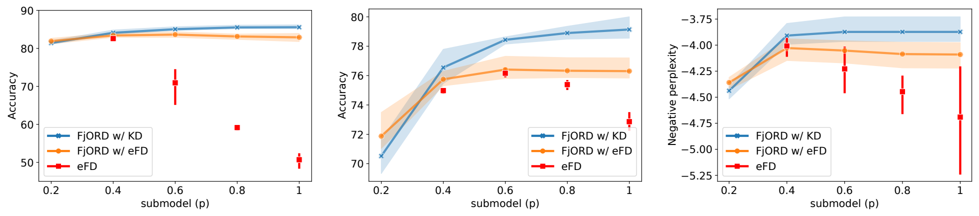
(a) Datasets description

	$p = 0.2$	0.4	0.6	0.8	1.0
CIFAR10 / ResNet18					
MACs	23M	91M	203M	360M	555M
Params	456K	2M	4M	7M	11M
FEMNIST / CNN					
MACs	47K	120K	218K	342K	491K
Params	5K	10K	15K	20K	26K
Shakespeare / RNN					
MACs	12K	40K	83K	143K	216K
Params	12K	40K	82K	142K	214K

(b) MACs and parameters per p-reduced network

5.1 Experimental Setup

Infrastructure. FjORD was implemented on top of the `Flower` (v0.14dev) [6] framework and PyTorch (v1.4.0) [54]. We run all our experiments on a private cloud cluster, consisting of Nvidia V100 GPUs. To scale to hundreds of clients on a single machine, we optimized `Flower` so that clients only allocate GPU resources when actively participating in a federated client round. We report average performance and the standard deviation across three runs for all experiments. To model client availability, we run up to 100 `Flower` clients in parallel and sample 10% at each global round, with the ability for clients to switch identity at the beginning of each round to overprovision for larger federated datasets. Furthermore, we model client heterogeneity by assigning each client to one of the device clusters. We provide the following setups:



(a) ResNet18 - CIFAR10 (b) CNN - FEMNIST (c) RNN - Shakespeare

Figure 4: Ordered Dropout with KD vs eFD baselines. Performance vs dropout rate p across different networks and datasets. $D_{\mathcal{P}} = \mathcal{U}_5$

Uniform-{5,10}: This refers to the distribution $D_{\mathcal{P}}$, *i.e.* $p \sim \mathcal{U}_k$, with $k = 5$ or 10.

Drop Scale $\in \{0.5, 1.0\}$**:** This parameter affects a possible skew in the number of devices per cluster. It refers to the drop in clients per cluster of devices, as we go to higher p's. Formally, for *uniform-n* and drop scale ds, the high-end cluster n contains $1-\sum_{i=0}^{n-1} {}^{ds}/_{n}$ of the devices and the rest of the clusters contain ${}^{ds}/_{n}$ each. Hence, for ds=1.0 of the *uniform-5* case, all devices can run the $p = 0.2$ subnetwork, 80% can run the $p = 0.4$ and so on, leading to a device distribution of $(0.2, ..., 0.2)$. This percentage drop is half for the case of ds=0.5, resulting in a larger high-end cluster, *e.g.* $(0.1, 0.1, ..., 0.6)$.

Baselines. To assess the performance against the state-of-the-art, we compare FjORD with the following baselines: i) Extended Federated Dropout (eFD), ii) FjORD with eFD (FjORD w/ eFD).

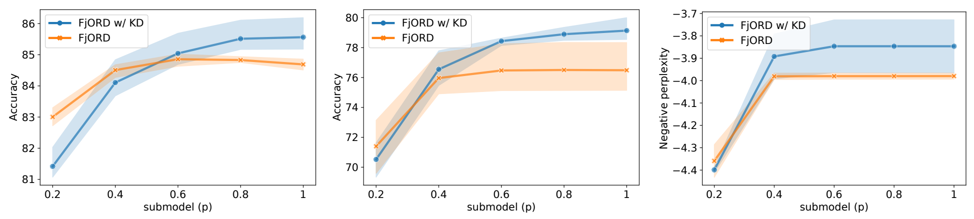
eFD builds on top of the technique of Federated Dropout (FD) [10], which adopts a Random Dropout (RD) at neuron/filter level for minimising the model's footprint. However, FD does not support adaptability to heterogeneous client capabilities out of the box, as it inherits a *single* dropout rate across devices. For this reason, we propose an extension to FD, allowing to adapt the dropout rate to the device capabilities, defined by the respective cluster membership. It is clear that eFD dominates FD in performance and provides a tougher baseline, as the latter needs to impose the same dropout rate to fit the model at hand on all devices, leading to larger dropout rates (*i.e.* uniform dropout of 80% for full model to support the low-end devices). We provide empirical evidence for this in the Appendix. For investigative purposes, we also applied eFD on top of FjORD, as a means to update a larger part of the model from lower-tier devices, *i.e.* allow them to evaluate submodels beyond their $p^i_{\max}$ during training.

5.2 Performance Evaluation

In order to evaluate the performance of FjORD, we compare it to the two baselines, eFD and OD+eFD. We consider the *uniform-5* setup with drop scale of 1.0 (*i.e.* uniform clusters). For each baseline, we train one independent model $\mathbf{F}_p$, end-to-end, for each p. For eFD, what this translates to is that the clusters of devices that cannot run model $\mathbf{F}_p$ compensate by randomly dropping out neurons/filters. We point out that $p = 0.2$ is omitted from the eFD results as it is essentially not employing any dropout whatsoever. For the case of FjORD + eFD, we control the RD by capping it to $d = 0.25$. This allows for larger submodels to be updated more often – as device belonging to cluster c can now have $p^c_{\max} \to p^{c+1}_{\max}$ during training where c+1 is the next more powerful cluster – while at the same time it prevents the destructive effect of too high dropout values shown in the eFD baseline.

Fig. 4 presents the achieved accuracy for varying values of p across the three target datasets. FjORD (denoted by FjORD w/ KD) outperforms eFD across all datasets with improvements between 1.53-34.87 percentage points (pp) (19.22 pp avg. across p values) on CIFAR10, 1.57-6.27 pp (3.41 pp avg.) on FEMNIST and 0.01-0.82 points (p) (0.46 p avg.) on Shakespeare. Compared to FjORD +eFD, FjORD achieves performance gains of 0.71-2.66 pp (1.79 avg.), up to 2.56 pp (1.35 pp avg.) on FEMNIST and 0.12-0.22 p (0.18 p avg.) on Shakespeare.

Across all tasks, we observe that FjORD is able to improve its performance with increasing p due to the nested structure of its OD method. We also conclude that eFD on top of FjORD does not seem to lead to better results. More importantly though, given the heterogeneous pool of devices, to obtain the highest performing model for eFD, multiple models have to be trained (*i.e.* one per device cluster). For instance, the highest performing models for eFD are $\mathbf{F}_{0.4}$, $\mathbf{F}_{0.6}$ and $\mathbf{F}_{0.4}$ for CIFAR10, FEMNIST and Shakespeare respectively, which can be obtained only *a posteriori*; after all model variants have been trained. Instead, despite the device heterogeneity, FjORD requires a single training process that leads to a global model that significantly outperforms the best model of eFD (by 2.98 and 2.73 pp for CIFAR10 and FEMNIST, respectively, and 0.13 p for Shakespeare), while allowing



(a) ResNet18 - CIFAR10 (b) CNN - FEMNIST (c) RNN - Shakespeare

Figure 5: Ablation analysis of FjORD with Knowledge Distillation. Ordered Dropout with $D_{\mathcal{P}} = \mathcal{U}_5$, KD - Knowledge distillation.

the direct, seamless extraction of submodels due to the nested structure of OD. Empirical evidence of the convergence of FjORD and the corresponding baselines is provided in the Appendix.

5.3 Ablation Study of KD in FjORD

To evaluate the contribution of our *knowledge distillation* method to the attainable performance of FjORD, we conduct an ablative analysis on all three datasets. We adopt the same setup of *uniform-5* and drop scale $= 1.0$ as in the previous section and compare FjORD with and without KD.

Fig. 5 shows the efficacy of FjORD's KD in FL settings. FjORD's KD consistently improves the performance across all three datasets when $p > 0.4$, with average gains of $0.18, 0.68$ and 0.87 pp for submodels of size $0.6, 0.8$ and 1 on CIFAR-10, $1.96, 2.39$ and 2.65 pp for FEMNIST and 0.10 p for Shakespeare. For the cases of $p \leq 0.4$, the impact of KD is fading. We believe this to be a side-effect of optimising for the average accuracy across submodels, which also yielded the $T = \alpha = 1$ strategy. We leave the exploration of alternative weighted KD strategies as future work. Overall, the use of KD significantly improves the performance of the global model, yielding gains of 0.71 and 2.63 pp for CIFAR10 and FEMNIST and 0.10 p for Shakespeare.

5.4 FjORD's Deployment Flexibility

5.4.1 Device Clusters Scalability

An important characteristic of FjORD is its ability to *scale* to a larger number of device clusters or, equivalently, perform well with higher granularity of p values. To illustrate this, we test the performance of OD across two setups, *uniform-5* and *-10* (defined in § 5.1).

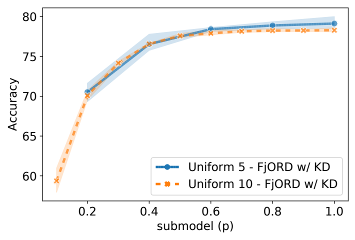
As shown in Fig. 6, FjORD sustains its performance even under the higher granularity of p values. This means that for applications where the modelling of clients needs to be more fine-grained, FjORD can still be of great value, without any significant degradation in achieved accuracy per submodel. This further supports the use-case where device-load needs to be modelled explicitly in device clusters (*e.g.* modelling device capabilities and load with deciles).

5.4.2 Adaptability to Device Distributions

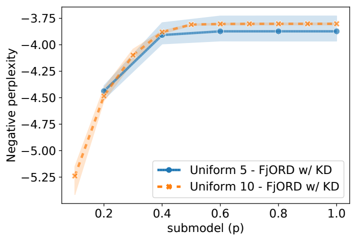
In this section, we make a similar case about FjORD's *elasticity* with respect to the allocation of available devices to each cluster. We adopt the setup of *uniform-5* once again, but compare across drop scales 0.5 and 1.0 (defined in § 5.1). In both cases, clients that can support models of $p^i_{\max} \in \{0.2, \ldots, 0.8\}$ are equisized, but the former halves the percentage of devices and allocates it to the last (high-end) cluster, now accounting for 60% of the devices. The rationale behind this is that the majority of participating devices are able to run the whole original model.
The results depicted in Fig. 7 show that the larger submodels are expectedly more accurate, being updated more often. However, the same graphs also indicate that FjORD does not significantly degrade the accuracy of the smaller submodels in the presence of more high-tier devices (*i.e.* $ds = 0.5$). This is a direct consequence of sampling p values during local rounds, instead of tying each tier with only the maximal submodel it can handle. We should also note that we did not alter the uniform sampling in this case on the premise that high-end devices are seen more often, precisely to illustrate FjORD's adaptability to latent user device distribution changes of which the server may not be aware.

6 Related Work

Dropout Techniques. Contrary to conventional Random Dropout [62], which stochastically drops a different, random set of a layer's units in every batch and is typically applied for regularisation purposes, OD employs a *structured* ordered dropping scheme that aims primarily at tunably reducing the computational and memory cost of training and inference. However, OD can still have an implicit regularisation effect since we encourage learning towards the top-ranked units (*e.g.* the left-most units in the example of Fig. 2), as these units will be dropped less often during training. Respectively,

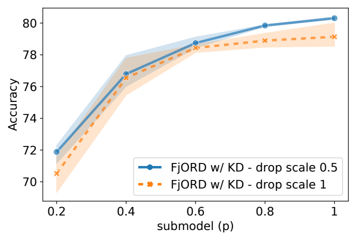


(a) CNN - FEMNIST

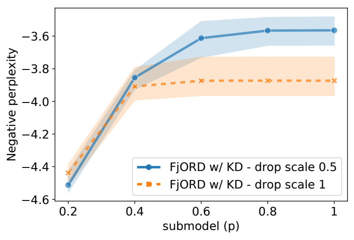


(b) RNN - Shakespeare

Figure 6: Demonstration of FjORD's scalability with respect to the number of device clusters.



(a) CNN - FEMNIST



(b) RNN - Shakespeare

Figure 7: Demonstration of the adaptability of FjORD across different device distributions.

at inference time, the load of a client can be dynamically adjusted by dropping the least important units, *i.e.* adjusting the width of the network.

To the best of our knowledge, the only similar technique to OD is Nested Dropout, where the authors proposed a similar construction, which is applied to the representation layer in autoencoders [57] in order to enforce identifiability of the learned representation or the last layer of the feature extractor [27] to learn an ordered set of features for transfer learning. In our case, we apply OD to every layer to elastically adapt the computation and memory requirements during training and inference.

Traditional Pruning. Conventional non-FL compression techniques can be applicable to reduce the network size and computation needs. The majority of pruning methods [21, 19, 40, 38, 52] aim to generate a *single* pruned model and require access to labelled data in order to perform a costly fine-tuning/calibration for *each* pruned variant. Instead, FjORD's Ordered Dropout enables the deterministic extraction of *multiple* pruned models with *varying* resource budgets directly after training. In this manner, we remove both the excessive overhead of fine-tuning and the need for labelled data availability, which is crucial for real-world, privacy-aware applications [63, 59]. Finally, other model compression methods [16, 67, 12] remain orthogonal to FjORD.

System Heterogeneity. So far, although substantial effort has been devoted to alleviating the *statistical heterogeneity* [41] among clients [61, 39, 29, 15, 43], the *system heterogeneity* has largely remained unaddressed. Considering the diversity of client devices, techniques on client selection [53] and control of the per-round number of participating clients and local iterations [48, 68] have been developed. Nevertheless, as these schemes are restricted to allocate a uniform amount of work to each selected client, they either limit the model complexity to fit the lowest-end devices or exclude slow clients altogether. From an aggregation viewpoint, [42] allows for partial results to be integrated to the global model, thus enabling the allocation of different amounts of work across heterogeneous clients. Despite the fact that each client is allowed to perform a different number of local iterations based on its resources, large models still cannot be accommodated on the more constrained devices.

Communication Optimisation. The majority of existing work has focused on tackling the communication overhead in FL. [35] proposed using structured and sketched updates to reduce the transmitted data. ATOMO [64] introduced a generalised gradient decomposition and sparsification technique, aiming to reduce the gradient sizes communicated upstream. [20] adaptively select the gradients' sparsification degree based on the available bandwidth and computational power. Building upon gradient quantisation methods [47, 26, 56, 28], [2] proposed using quantisation in the model sharing and aggregation steps. However, their scheme requires the *same* clients to participate across all rounds, and is, thus, unsuitable for realistic settings where clients' availability cannot be guaranteed. Despite the bandwidth savings, these communication-optimising approaches do not offer computational gains nor do they address device heterogeneity. Nonetheless, they remain orthogonal to our work and can be complementarily combined to further alleviate the communication cost.

Computation-Communication Co-optimisation. A few works aim to co-optimise both the computational and bandwidth costs. PruneFL [32] proposes an unstructured pruning method. Despite the similarity to our work in terms of pruning, this method assumes a *common* pruned model across *all* clients at a given round, thus not allowing more powerful devices to update more weights. Hence, the pruned model needs to meet the constraints of the least capable devices, which severely limits the model capacity. Moreover, the adopted unstructured sparsity is difficult to translate to processing speed gains [70]. Federated Dropout [10] randomly sparsifies the global model, before sharing it to the clients. Similarly to PruneFL, Federated Dropout does not consider the system diversity and distributes the *same* model size to all clients. Thus, it is restricted by the low-end devices or excludes them altogether from the FL process. Additionally, Federated Dropout does not translate to computational benefits at inference time, since the whole model is deployed after federated training.

Contrary to the presented works, our framework embraces the client heterogeneity, instead of treating it as a limitation, and thus pushes the boundaries of FL deployment in terms of fairness, scalability and performance by tailoring the model size to the device at hand, both at training and inference time, in a "train-once-deploy-everywhere" manner.

7 Conclusions & Future Work

In this work, we have introduced FjORD, a federated learning method for heterogeneous device training. To this direction, FjORD builds on top of our Ordered Dropout technique as a means to extract submodels of smaller footprints from a main model in a way where training the part also participates in training the whole. We show that our Ordered Dropout is equivalent to SVD for linear mappings and demonstrate that FjORD's performance in the local and federated setting exceeds that of competing techniques, while maintaining flexibility across different environment setups.

In the future, we plan to investigate how FjORD can be deployed and extended to future-gen devices and models in a life-long manner, the interplay between system and data heterogeneity for OD-based personalisation as well as alternative dynamic inference techniques for tackling system heterogeneity.

Broader Impact

Our work has a dual broader societal impact: i) on privacy and fairness in participation and ii) on the environment. On the one hand, centralised DNN training [22] has been the norm for a long time, mainly facilitated by the advances in server-grade accelerator design and cheap storage. However, this paradigm comes with a set of disadvantages, both in terms of data privacy and energy consumption. With mobile and embedded devices becoming more capable and FL becoming a viable alternative [3, 8], one can leverage the free compute cycles of client devices to train models on-device, without data ever leaving the device premises. These devices, being typically battery-powered, operate under a more constrained power envelope compared to data-center accelerators [1]. Moreover, these devices are already deployed in the wild, but typically not used for training purposes. What FjORD contributes is the ability for even less capable devices to participate in the training process, thus increasing the representation of low-tier devices (and by extension the correlated demographic groups), as well as adding to the overall compute capabilities of the distributed system as a whole, potentially offsetting part of the carbon footprint of centralised training data centers [55].

However, moving the computation cost from the service provider to the user of the device is a non-negligible step and the user should be made aware what their device is used for, especially if they are contributing to the knowledge of a model they do not own. Moreover, while many large data centers [14, 18] are increasingly dependent on renewable resources for meeting their power demands, this might not be the case for household electricity, which may impede the sustainability of training on device, at least in the short run.

Funding Disclosure

This work was entirely performed at and funded by Samsung AI.

References

[1] Mario Almeida, Stefanos Laskaridis, Ilias Leontiadis, Stylianos I. Venieris, and Nicholas D. Lane. EmBench: Quantifying Performance Variations of Deep Neural Networks Across Modern Commodity Devices. In *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications (EMDL)*, 2019.

[2] Mohammad Mohammadi Amiri, Deniz Gunduz, Sanjeev R Kulkarni, and H Vincent Poor. Federated Learning with Quantized Global Model Updates. *arXiv preprint arXiv:2006.10672*, 2020.

[3] Apple. Learning with Privacy at Scale. In *Differential Privacy Team Technical Report*, 2017.

[4] Authors. TensorFlow Federated Datasets. 2019.

[5] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How To Backdoor Federated Learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 2938–2948, 2020.

[6] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D Lane. Flower: A Friendly Federated Learning Research Framework. *arXiv preprint arXiv:2007.14390*, 2020.

[7] Keith Bonawitz et al. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[8] Keith Bonawitz et al. Towards Federated Learning at Scale: System Design. In *Proceedings of Machine Learning and Systems (MLSys)*, 2019.

[9] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečǹy, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[10] Sebastian Caldas, Jakub Konečný, Brendan McMahan, and Ameet Talwalkar. Expanding the Reach of Federated Learning by Reducing Client Resource Requirements. In *NeurIPS Workshop on Federated Learning for Data Privacy and Confidentiality*, 2018.

[11] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[12] Łukasz Dudziak, Mohamed S Abdelfattah, Ravichander Vipperla, Stefanos Laskaridis, and Nicholas D Lane. ShrinkML: End-to-End ASR Model Compression Using Reinforcement Learning. In *INTERSPEECH*, pages 2235–2239, 2019.

[13] European Commission. GDPR: 2018 Reform of EU Data Protection Rules.

[14] Facebook. Software, servers, systems, sensors, and science: Facebook's recipe for hyperefficient data centers. `https://tech.fb.com/hyperefficient-data-centers/`, 2021. Accessed: January 10, 2022.

[15] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized Federated Learning with Theoretical Guarantees: A Model-Agnostic Meta-Learning Approach. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[16] Biyi Fang, Xiao Zeng, and Mi Zhang. NestDNN: Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 115–127, 2018.

[17] Robin C. Geyer, Tassilo J. Klein, and Moin Nabi. Differentially Private Federated Learning: A Client Level Perspective. In *NeurIPS Workshop on Machine Learning on the Phone and other Consumer Devices (MLPCD)*, 2017.

[18] Google. Google datacenters efficiency. `https://www.google.co.uk/about/datacenters/efficiency/`, 2021. Accessed: January 10, 2022.

[19] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic Network Surgery for Efficient DNNs. In *Advances in Neural Information Processing Systems (NeuriPS)*, pages 1387–1395, 2016.

[20] Pengchao Han, Shiqiang Wang, and Kin K Leung. Adaptive Gradient Sparsification for Efficient Federated Learning: An Online Learning Approach. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2020.

[21] Song Han, Jeff Pool, John Tran, and William Dally. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1135–1143, 2015.

[22] K. Hazelwood et al. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[24] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. In *NeurIPS Deep Learning Workshop*, 2014.

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[26] Samuel Horváth, Chen-Yu Ho, L'udovít Horváth, Atal Narayan Sahu, Marco Canini, and Peter Richtárik. Natural Compression for Distributed Deep Learning. *arXiv preprint arXiv:1905.10988*, 2019.

[27] Samuel Horváth, Aaron Klein, Peter Richtárik, and Cédric Archambeau. Hyperparameter transfer learning with adaptive complexity. In *International Conference on Artificial Intelligence and Statistics*, pages 1378–1386. PMLR, 2021.

[28] Samuel Horváth and Peter Richtárik. A Better Alternative to Error Feedback for Communication-Efficient Distributed Learning. In *International Conference on Learning Representations*, 2021.

[29] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip Gibbons. The Non-IID Data Quagmire of Decentralized Machine Learning. In *International Conference on Machine Learning (ICML)*, 2020.

[30] R. Hu, Y. Guo, H. Li, Q. Pei, and Y. Gong. Personalized Federated Learning With Differential Privacy. *IEEE Internet of Things Journal (JIOT)*, 7(10):9530–9539, 2020.

[31] Andrey Ignatov, Radu Timofte, Andrei Kulik, Seungsoo Yang, Ke Wang, Felix Baum, Max Wu, Lirong Xu, and Luc Van Gool. AI Benchmark: All About Deep Learning on Smartphones in 2019. In *International Conference on Computer Vision Workshops (ICCVW)*, 2019.

[32] Yuang Jiang, Shiqiang Wang, Bong Jun Ko, Wei-Han Lee, and Leandros Tassiulas. Model Pruning Enables Efficient Federated Learning on Edge Devices. In *Workshop on Scalability, Privacy, and Security in Federated Learning (SpicyFL), NeurIPS*, 2020.

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

[34] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In *International Conference on Machine Learning (ICML)*, 2020.

[35] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated Learning: Strategies for Improving Communication Efficiency. In *NeurIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[36] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[37] Stefanos Laskaridis, Stylianos I. Venieris, Hyeji Kim, and Nicholas D. Lane. HAPI: Hardware-Aware Progressive Inference. In *International Conference on Computer-Aided Design (ICCAD)*, 2020.

[38] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: Single-Shot Network Pruning based on Connection Sensitivity. In *International Conference on Learning Representations (ICLR)*, 2019.

[39] Daliang Li and Junpu Wang. FedMD: Heterogenous Federated Learning via Model Distillation. In *NeurIPS 2019 Workshop on Federated Learning for Data Privacy and Confidentiality*, 2019.

[40] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient ConvNets. In *International Conference on Learning Representations (ICLR)*, 2016.

[41] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine*, 2020.

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

[43] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair Resource Allocation in Federated Learning. In *International Conference on Learning Representations (ICLR)*, 2020.

[44] Xiaoxiao Li, Meirui JIANG, Xiaofei Zhang, Michael Kamp, and Qi Dou. Fed{BN}: Federated Learning on Non-{IID} Features via Local Batch Normalization. In *International Conference on Learning Representations (ICLR)*, 2021.

[45] Paul Pu Liang, Terrance Liu, Liu Ziyin, Nicholas B Allen, Randy P Auerbach, David Brent, Ruslan Salakhutdinov, and Louis-Philippe Morency. Think Locally, Act Globally: Federated Learning with Local and Global Representations. In *NeurIPS 2019 Workshop on Federated Learning*, 2019.

[46] Robert LiKamWa and Lin Zhong. Starfish: Efficient Concurrency Support for Computer Vision Applications. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 213–226, 2015.

[47] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. In *International Conference on Learning Representations (ICLR)*, 2018.

[48] Bing Luo, Xiang Li, Shiqiang Wang, Jianwei Huang, and Leandros Tassiulas. Cost-Effective Federated Learning Design. In *INFOCOM*, 2021.

[49] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[50] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning Differentially Private Recurrent Language Models. In *International Conference on Learning Representations (ICLR)*, 2018.

[51] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting Unintended Feature Leakage in Collaborative Learning. In *IEEE Symposium on Security and Privacy (SP)*, pages 691–706, 2019.

[52] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance Estimation for Neural Network Pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11264–11272, 2019.

[53] Takayuki Nishio and Ryo Yonetani. Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge. In *IEEE International Conference on Communications (ICC)*, 2019.

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8026–8037, 2019.

[55] Xinchi Qiu, Titouan Parcollet, Daniel J. Beutel, Taner Topal, Akhil Mathur, and Nicholas D. Lane. A first look into the carbon footprint of federated learning. *CoRR*, abs/2010.06537, 2020.

[56] Aditya Rajagopal, Diederik Vink, Stylianos Venieris, and Christos-Savvas Bouganis. Multi-Precision Policy Enforced Training (MuPPET) : A Precision-Switching Strategy for Quantised Fixed-Point Training of CNNs. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 7943–7952, 2020.

[57] Oren Rippel, Michael Gelbart, and Ryan Adams. Learning Ordered Representations with Nested Dropout. In *International Conference on Machine Learning (ICML)*, pages 1746–1754, 2014.

[58] F. Sattler, S. Wiedemann, K. R. Müller, and W. Samek. Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 31(9):3400–3413, 2020.

[59] Reza Shokri and Vitaly Shmatikov. Privacy-Preserving Deep Learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1310–1321, 2015.

[60] Sidak Pal Singh and Martin Jaggi. Model Fusion via Optimal Transport. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 2020.

[61] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated Multi-Task Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[62] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research (JMLR)*, 15(56):1929–1958, 2014.

[63] Martin J Wainwright, Michael Jordan, and John C Duchi. Privacy Aware Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[64] Hongyi Wang, Scott Sievert, Shengchao Liu, Zachary Charles, Dimitris Papailiopoulos, and Stephen Wright. ATOMO: Communication-Efficient Learning via Atomic Sparsification. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[65] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated Learning with Matched Averaging. In *International Conference on Learning Representations (ICLR)*, 2020.

[66] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[67] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-Aware Automated Quantization with Mixed Precision. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8612–8620, 2019.

[68] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive Federated Learning in Resource Constrained Edge Computing Systems. *IEEE Journal on Selected Areas in Communications (JSAC)*, 37(6), 2019.

[69] C. Wu et al. Machine Learning at Facebook: Understanding Inference at the Edge. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[70] Zhuliang Yao, Shijie Cao, Wencong Xiao, Chen Zhang, and Lanshun Nie. Balanced Sparsity for Efficient DNN Inference on GPU. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 5676–5683, 2019.

[71] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. Bayesian Nonparametric Federated Learning of Neural Networks. In *International Conference on Machine Learning (ICML)*, pages 7252–7261. PMLR, 2019.

[72] Barret Zoph and Quoc Le. Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Supplementary Material

A Proof of Theorem 1

Proof. Let $A = \hat{U}\Sigma\hat{V}^\top = \sum_{i=1}^{k} \sigma_i \hat{u}_i \hat{v}_i^\top$ denote SVD decomposition of A. This decomposition is unique as A is full rank with distinct singular values. We also denote $A_b = \sum_{i=1}^{b} \sigma_i \hat{u}_i \hat{v}_i^\top$

Assuming a linkage of input data x and response y through a linear mapping $Ax = y$, we obtain the following

$$\min_{U,V} \mathbb{E}_{x\sim\mathcal{X}} \mathbb{E}_{p\sim D_\mathcal{P}} \|\mathbf{F}_p(x) - y\|^2 = \min_{U,V} \mathbb{E}_{x\sim\mathcal{X}} \mathbb{E}_{p\sim D_\mathcal{P}} \|\mathbf{F}_p(x) - Ax\|^2.$$

Let us denote u_i to be i-th row of matrix U and $v_i^\top$ to be i-th row of V. Due to $\mathcal{X}$ being uniform on unit ball and structure of the neural network, we can further simplify the objective to

$$\min_{U,V} \mathbb{E}_{p\sim D_\mathcal{P}} \left\| \sum_{i=1}^{\lceil p\cdot k\rceil} u_i v_i^\top - A \right\|_F^2,$$

where F denotes Frobenius norm. Since for each b there exists nonzero probability $\mathbf{P}_b$ such that $b = \lceil p \cdot k \rceil$, we can explicitly compute expectation, which leads to

$$\min_{U,V} \sum_{b=1}^{k} \mathbf{P}_b \left\| \sum_{i=1}^{b} u_i v_i^\top - A \right\|_F^2.$$

Realising that $\sum_{i=1}^{b} u_i v_i^\top$ has rank at most b, we can use Eckart–Young theorem which implies that

$$\min_{U,V} \sum_{b=1}^{k} \mathbf{P}_b \left\| \sum_{i=1}^{b} u_i v_i^\top - A \right\|_F^2 \geq \sum_{b=1}^{k} \mathbf{P}_b \|A_b - A\|_F^2.$$

Equality is obtained if and only if $A_b = \sum_{i=1}^{b} u_i v_i^\top$ for all $i \in \{1, 2, \ldots, k\}$. This can be achieved, *e.g.* $v_i = \hat{v}_i$ and $u_i = \sigma_i \hat{u}_i$ for all $i \in \{1, 2, \ldots, k\}$. □

B OD: Optimization Perspective

In this section, we discuss the impact of introducing the Ordered Dropout formulation into the original problem. We follow notation used in the main paper, where $\mathbf{F}$ and $\boldsymbol{w}$ are the original network and weights, respectively, and $\mathbf{F}_p$ denotes a pruned p-subnetwork with its weights $\boldsymbol{w}_p$. We argue that the problem does not become harder from the optimization point of view as quantities such as smoothness or strong convexity do not worsen for the OD formulation as stated in the following lemma.

Lemma 1. *Let* $\mathbf{F}$ *be* μ*-strongly convex and* L*-smooth. Then* $\mathbf{F}_{D_\mathcal{P}} = \mathbb{E}_{p\sim D_\mathcal{P}}[\mathbf{F}_p]$ *is* μ'*-strongly convex and* L'*-smooth with* $\mu' \geq \mu$ *and* $L' \leq L$.

Proof. The claim trivially follows from the definitions of smoothness and strong convexity by realising that $\mathbf{F}_p(\boldsymbol{w}_p)$ is equal to $\mathbf{F}(\boldsymbol{w})$ where $\boldsymbol{w}_p$ is obtained from $\boldsymbol{w}$ using our pruning technique, thus $\mathbf{F}_p$ is μ_p-strongly convex and L_p-smooth with $\mu_p \geq \mu$ and $L_p \leq L$ for all $p \in [0, 1]$. Subsequently, the same has to hold for the expectation. □

C Experimental Details

C.1 Datasets and Models

Below, we provide detailed description of the datasets and models used in this paper. We use vision datasets EMNIST [11] and its federated equivalent FEMNIST and CIFAR10 [36], as well as the language modelling dataset Shakespeare [49]. In the centralised training scenarios, we use the union of dataset partitions for training and validation, while in the federeated deployment, we adopt either a random partitioning in IID datasets or the pre-partitioned scheme available in TensorFlow Federated (TFF) [4]. Detailed description of the datasets is provided below.

CIFAR10. The CIFAR10 dataset is a computer vision dataset consisting of $32\times32\times3$ images with 10 possible labels. For federated version of CIFAR10, we randomly partition dataset among 100 clients, each client having 500 data-points. We train a ResNet18 [23] on this dataset, where for Ordered Dropout, we train independent batch normalization layers for every $p \in D_{\mathcal{P}}$ as Ordered Dropout affects distribution of layers' outputs. We perform standard data augmentation and preprocessing, *i.e.* a random crop to shape $(24, 24, 3)$ followed by a random horizontal flip and then we normalize the pixel values according to their mean and standard deviation.

(F)EMNIST. EMNIST consists of 28×28 gray-scale images of both numbers and upper and lower-case English characters, with 62 possible labels in total. The digits are partitioned according to their author, resulting in a naturally heterogeneous federated dataset. EMNIST is collection of all the data-points. We do not use any preprocessing on the images. We train a Convolutional Neural Network (CNN), which contains two convolutional layers, each with 5×5 kernels with 10 and 20 filters, respectively. Each convolutional layer is followed by a 2×2 max pooling layer. Finally, the model has a dense output layer followed by a softmax activation. FEMNIST refers to the federated variant of the dataset, which has been partitioned based on the writer of the digit/character [9].

Shakespeare. Shakespeare dataset is also derived from the benchmark designed by [9]. The dataset corpus is the collected works of William Shakespeare, and the clients correspond to roles in Shakespeare's plays with at least two lines of dialogue. Non-federated dataset is constructed as a collection of all the clients' data-points in the same way as for FEMNIST. For the preprocessing step, we apply the same technique as TFF dataloader, where we split each client's lines into sequences of 80 characters, padding if necessary. We use a vocabulary size of 90 entities – 86 characters contained in Shakespeare's work, beginning and end of line tokens, padding tokens, and out-of-vocabulary tokens. We perform next-character prediction on the clients' dialogue using an Recurrent Neural Network (RNN). The RNN takes as input a sequence of 80 characters, embeds it into a learned 8-dimensional space, and passes the embedding through two LSTM [25] layers, each with 128 units. Finally, we use a softmax output layer with 90 units. For this dataset, we don't apply Ordered Dropout to the embedding layer, but only to the subsequent LSTM layers, due to its insignificant impact on the size of the model.

C.2 Implementation Details

FjORD was built on top of `PyTorch` [54] and `Flower` [6], an open-source federated learning framework which we extended to support Ordered, Federated, and Adaptive Dropout and Knowledge Distillation. Our OD aggregation was implemented in the form of a `Flower` strategy that considers each client maximum width $p^i_{\max}$. Server and clients run in a multiprocess setup, communicating over gRPC[7] channels and can be distributed across multiple devices. To scale to hundreds of clients per cloud node, we optimised `Flower` so that clients only allocate GPU resources when actively participating in a federated client round. This is accomplished by separating the forward/backward propagation of clients into a separate spawned process which frees its resources when finished. Timeouts are also introduced in order to limit the effect of stragglers or failed client processes to the entire training round.

C.3 Hyperparameters.

In this section we lay out the hyperparameters used for each $\langle$model, dataset, deployment$\rangle$ tuple.

C.3.1 Non-federated Experiments

For centralised training experiments, we employ `SGD` with momentum 0.9 as an optimiser. We also note that the training epochs of this setting are significantly fewer that the equivalent federated training rounds, as each iteration is a full pass over the dataset, compared to an iteration over the sampled clients.

[7]`https://www.grpc.io/`

ResNet18. We use batch size 128, step size of 0.1 and train for 300 epochs. We decrease the step size by a factor of 10 at epochs 150 and 225.
CNN. We use batch size 128 and train for 20 epochs. We keep the step size constant at 0.1.
RNN. We use batch size 32 and train for 50 epochs. We keep the step size constant at 0.1.

C.3.2 Federated Experiments

For each ⟨model, dataset⟩ federated deployment, we start the communication round by randomly sampling 10 clients to model client availability and for each available client we run one local epoch. We decrease the client step size by 10 at 50% and 75% of total rounds. We run 500 global rounds of training across experiments and use `SGD` without momentum.

ResNet18. We use local batch size 32 and step size of 0.1.
CNN. We use local batch size 16 and step size of 0.1.
RNN. We use local batch size 4 and step size of 1.0.

D Additional Experiments

Ordered Dropout exactly recovers SVD: Empirical evidence

In this experiment, we want to empirically back up our theoretical claims about Ordered Dropout recovering SVD. To this end, we generate a normal random 5×5 matrix, compute its SVD (USV^T) and set $A = UDV^T$ where D is a diagonal matrix with $5, 4, \ldots, 1$ on the diagonal to ensure full rank with distinct singular values. We initialize U, V (1st & 2nd layer, see Theorem 1's proof) as normal random matrices. We then run SGD with lr=0.1 for 10k iterations. In each step, we sample 32 points from the 5D unit ball and apply OD sampled from the uniform distribution, taking an SGD step. In Fig. 8, we display the Frobenius norm between the best k-rank approximation of A and the corresponding OD subnetwork ($p = [0.2, \ldots, 1]$, for $k = 1, \ldots, 5$). This converges and recovers SVD.

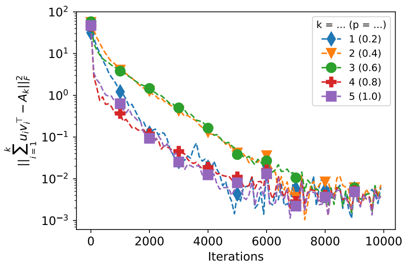
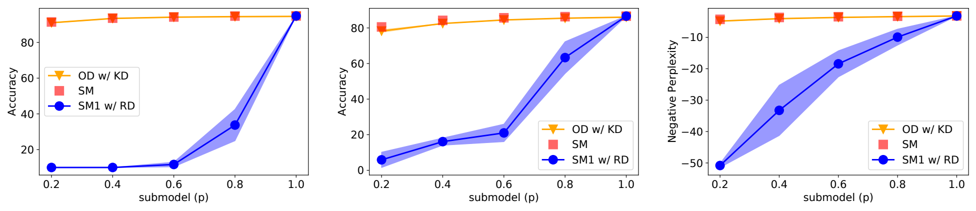


Figure 8: Frobenius norm of best k-rank approximation vs. corresponding OD sub-network.

Local training: Ordered Dropout vs. Random Dropout vs. Single Models

In this section, we present an extended version of Fig. 3, depicted in Fig. 9. We expand the comparison of OD against Random Dropout, where we pick the original model (SM1) and utilise different dropout rates for obtaining submodels. It can be easily witnessed that performance drops catastrophically in the latter case.

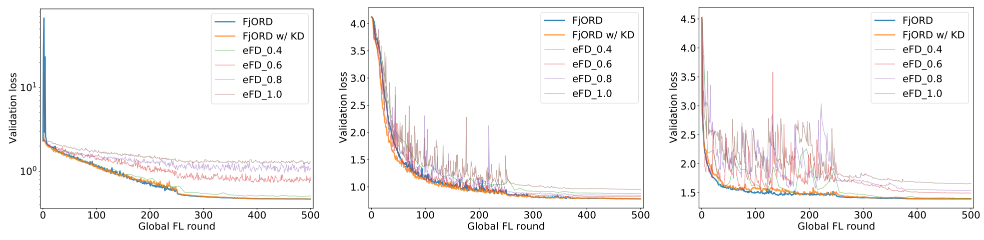


(a) ResNet18 - CIFAR10 (b) CNN - EMNIST (c) RNN - Shakespeare

Figure 9: Full non-federated datasets. OD-Ordered Dropout with $D_{\mathcal{P}} = \mathcal{U}_5$, SM-single independent models, KD-knowledge distillation, SM1 w/ RD-end-to-end trained full model with submodels obtained using Random Dropout instead Ordered Dropout.

Convergence of FjORD

In this section, we depict the convergence behaviour of FjORD compared to the eFD baseline across 500 global training rounds. We follow the same setup as in § 5.2. From Fig. 10, it can be easily witnessed that FjORD (with or without distillation) leads to smoother convergence and yields lower losses across the three (model,dataset) combinations.



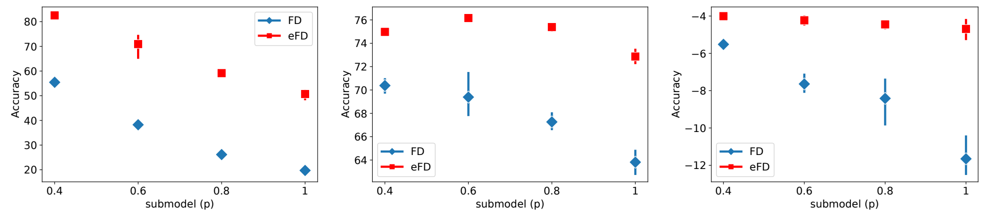
(a) ResNet18 - CIFAR10 (b) CNN - FEMNIST (c) RNN - Shakespeare

Figure 10: Convergence of FjORD vs. eFD over 500 FL training rounds.

Federated Dropout vs. eFD

In this section we provide evidence of eFD's accuracy dominance over FD. We inherit the setup of the experiment in § 5.2 to be able to compare results and extrapolate across similar conditions. From Fig. 11, it is clear that eFD's performance dominates the baseline FD by 27.13-33 percentage points (pp) (30.94 pp avg.) on CIFAR10, 4.59-9.04 pp (7.13 pp avg.) on FEMNIST and 1.51-6.96 points (p) (3.96 p avg.) on Shakespeare. The superior performance of eFD, as a technique, can be attributed to the fact that it allows for an adaptive dropout rate based on the device capabilities. As such, instead of imposing a uniformly high dropout rate to accommodate the low-end of the device spectrum, more capable devices are able to update larger portion of the network, thus utilising its capacity more intelligently.

However, it should be also noted that despite FD's accuracy drop, on average it is expected to have a lower computation/upstream network bandwidth/energy impact on devices of the higher end of the spectrum, as they run the largest dropout rate possible to accommodate the computational need of their lower-end counterparts. This behaviour, however, can also be interpreted as wasted computation potential on the higher end – especially under unconstrained environments (*i.e.* device charging overnight) – at the expense of global model accuracy.



(a) ResNet18 - CIFAR10 (b) CNN - FEMNIST (c) RNN - Shakespeare

Figure 11: Federated Dropout (FD) vs Extended Federated Dropout (eFD). Performance vs dropout rate p across different networks and datasets.

E Limitations

In this work we have presented a method for training DNNs in centralised (through OD) and federated (through FjORD) settings. We have evaluated our method in terms of accuracy performance against baselines and enhancements of those that represent the state-of-the-art at the time of writing across three different datasets and tasks in IID and non-IID settings. Our evaluation has adopted uniform sampling of p values and considered to be constant across layers of the network at hand mainly for simplicity and tractability of the problem. However, this does not degrade the generality of our approach.

While we do target heterogeneous devices found in the wild, such as mobile phones, we have not measured the performance of our technique on such devices, mainly due to the lack of maturity in tools for on-device training. However, we have demonstrated the performance gains in terms of FLOPs and parameters in Table 1b, which are directly correlated with on-device performance, memory footprint and communication size. We defer on-device benchmarking and in-the-wild deployment at scale for future work.

Last, we have assumed the clusters of devices to be given and the different device load to be assumed in the modelling of these clusters. While we do provide results across different number and distribution of clusters in § 5.4, we treat the device-to-cluster association outside the scope of this work.