20 RASPBERRY PI'S, ONE MODEL: FEDERATED LEARNING ON REAL HARDWARE

Søren Winkel Holm, Asger Laurits Schultz, Gustav Lang Moesmand

Technical University of Denmark

ABSTRACT

Federated Learning (FL) is emerging as an essential mechanism for assuring user privacy in large-scale machine learning (ML) [1]. Important use-cases of this learning paradigm run on a federation of real-world user devices, but in the literature, FL is often simulated in an artificial computing cluster environment [1, 2, 3]. Seeking to capture the unique FL problems and trade-offs when running on physical hardware, we set up 20 Raspberry Pi's acting as user devices. Using this experimental setup, we perform an empirical study of the influence of key hyperparameters of the FedAvg [2] algorithm. In the case of the number of local epochs on clients, the physical timings allow us to identify a trade-off between time spent on communication and local computation. Testing robustness against imbalanced data across the clients and noisy data, we highlight the potential of using stronger aggregation schemes than weight averaging by implementing the FedDF [3] algorithm.

1. INTRODUCTION

Large-scale surveys have shown that the growing use of Artificial Intelligence (AI) has resulted in a widespread fear of loss of personal privacy [4, 5]. As part of a general push towards safer AI, large tech companies such as Google and Apple have employed FL methods in cases including Siri, Google Chrome, and Gboard [1].

The term FL covers distributed ML setups where multiple clients collaborate in learning from local datasets which are not exchanged and where a central server aggregates local updates [1, 2]. Aggregation by iteratively averaging weights from models, each produced by training for several local epochs on each client, is known as the formative *FedAvg* [2] algorithm. For faster and more stable convergence, additional aggregation methods have been developed, including the ensemble distillation algorithm *FedDF* [3]. We refer to Kairouz et al. for an overview of the field [1].

Across this rich literature, many benchmarks of FL performance over algorithmic choices exist but are often performed by simulating the federation on central compute clusters [3]. In this project, we seek to capture the unique hardware setup of FL use-cases such as smartphones, where a number of computationally weak edge devices hold the data. This is achieved by performing local training of a convolutional neural network (CNN) on 20 Raspberry Pi devices over which CIFAR-10 [6] is divided and aggregating these locally trained models centrally using FedAvg. The project aims to investigate the impact of FedAvg hyperparameters, including the number of clients per communication round and local epochs on convergence time, analyse aggregation robustness against imbalanced and noisy data, and uncover performance trade-offs in the physical hardware setting.

2. METHODS

2.1. FL Methods

We implemented FL by setting up K clients each with a disjoint training dataset partition of size n_k. A global model, $\mathcal{M}_G$, was initialised and maintained on the central server, and for L rounds, dubbed *communication rounds*, $S \leq K$ clients were sampled, each receiving a copy of $\mathcal{M}_G$. Each client performed E local epochs of learning before returning the updated local model $\mathcal{M}_k$ to the server.

In FedAvg, the server aggregated the S returned models by averaging over all model weights, yielding a new global model to be sent out for the next communication round, minimizing an implicit objective function f of model weights $\mathbf{w}$ when using loss function ℓ:

$$f(\mathbf{w}) = \sum_{k=1}^{K} \frac{n_k}{n_{\text{total}}} F_k(\mathbf{w}), \quad F_k(\mathbf{w}) = \frac{1}{n_k} \sum_{i=1}^{n_k} \ell(x_i, y_i; \mathbf{w}),$$

in which $F_k(\mathbf{w})$ is the objective of the k'th client [2].

For an alternative aggregation method, we implemented the FedDF algorithm, where the FedAvg's weight averaging is replaced by running ensemble distillation for model fusion using an unlabeled dataset similar to the training datasets [3]. Copying the hyperparameters from [3], $\mathcal{M}_G^{(l+1)}$ was produced by distilling the S local models $\mathcal{M}_{k_i}^{(l)}$ for 10^4 batch updates against a Kulback-Leibler (KL) divergence criterion with a batch size of 128 and a learning rate of 10^{-3} used for Adam optimization with cosine annealing [7, 8]. Early stopping was implemented by calculating the KL divergence against an unlabeled validation set every 10^3 updates and terminating if evaluation loss did not fall (further described in Appendix C). The CIFAR-100 dataset was used for distillation [6].

2.2. Physical Devices

The project setup, shown in Figure 1, was divided into two parts A central high-performance cluster (HPC) and 20 Raspberry Pi 3B's, each with 1 GB memory and a quad-core 1.2 GHz CPU. Crucially, the Pi's were located on a network separate from the HPC to simulate a more realistic communication overhead. The HPC server was responsible for aggregating the local models trained by the Pi's and evaluating the resulting global model $\mathcal{M}_G$. The code was designed not to require physical devices, so experiments without hardware timing were run on an NVIDIA A100 for faster training time and a reduced power bill.

Every Pi ran a Flask server that transferred models, ran local training and broadcast running memory usage; an important consideration when running on such resource-limited devices. The Flask server also had a route for sending commands to allow primitive over-the-air-update functionality.

The Raspberry Pi setup was able to run experiments where $K > 20$, that is, maintaining more than 20 clients and thus dataset partitions with the constraint of no more than 20 clients being sampled each round: $S \leq 20$. This was done by storing all K client datasets on every device, and in each communication round l, assigning each of the S sampled client datasets to a Raspberry Pi.

The Pi's were connected to a switch which was connected via cable to the router. When turning off the switch, the Pi's were set to connect via Wi-Fi instead, allowing tests of the impact of communication overhead in two cases: under relatively fast Ethernet and relatively slow Wi-Fi. For reference, the network used had a bandwidth of 100/100 Mbit/s, all of which was utilised on Ethernet, but only about 40 % on Wi-Fi.

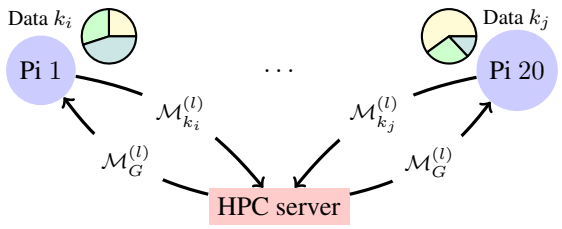

Fig. 1. Our federated setup performing updates at communication round l, at which $S \leq 20$ clients are sampled, each corresponding to a dataset partition $k_1 \ldots k_S$. Raspberry Pi 1 trains a model as client k_i, and Raspberry Pi 20 as k_j.

2.3. Deep Learning Problem

As an example learning problem, we chose the CIFAR-10 computer vision (CV) task of classifying 32×32 images into object classes such as birds, cats, and aeroplanes [6]. Due to device memory limits, all images were greyscaled. The training dataset contains 5K images for each of the 10 label.

For the model $\mathcal{M}$, we chose a network with two convolutional layers followed by two linear layers, which is further detailed in Appendix A. The model was limited in size to accommodate the strict memory limits on the Raspberry Pi's.

Optimization for the E local epochs on each device was performed by using the Adam optimizer [8] with a learning rate η, which was decayed every local epoch: $\eta \leftarrow \gamma\eta, \gamma \leq 1$.

2.4. Data Imbalance and Noise

2.4.1. Dirichlet sampling

The total training dataset was divided into K evenly sized partitions among all the clients. In practice, dataset class balance can rarely be assumed in the FL setting [1]. In order to simulate varying levels of imbalance, the Dirichlet distribution, $\mathrm{Dir}(\boldsymbol{\alpha})$, was used. The length of the parameter vector $\boldsymbol{\alpha}$ corresponds to the number of labels, 10, and we let $\alpha_i = \alpha$. Every sample $\boldsymbol{\pi} \sim \mathrm{Dir}(\boldsymbol{\alpha})$ is a probability distribution over labels and α determines the uniformity of this distribution. For $\alpha \to 0$, one label dominates, where as for $\alpha \to \infty$, $\boldsymbol{\pi}$ will be increasingly uniform, as exemplified in Appendix B.1. For $\alpha = 1$, every possible $\boldsymbol{\pi}$ is equally likely.

$\boldsymbol{\pi}$ was sampled for every client, making the label distribution Dirichlet for every client. To keep the client datasets disjoint while using a maximal part of the total dataset, we created a client-balancing Dirichlet sampling algorithm described in detail in Appendix B.2.

As a comparative baseline, we also simulate random partitions, denoted iid. following literature [3].

2.4.2. Noisy data

To simulate the fact that some user devices can be unreliable, we tested the concept of noisy clients. The training data on a noisy client had all labels replaced with randomly chosen classes, removing all signal. We tested performance over the number of noisy clients $N_K \leq K$ to simulate erroneous or even adversarial clients.

2.5. Evaluation

We performed experiments using FedAvg testing the effects of four variables: The number of clients sampled (S), the class balance (α), the number of local epochs (E) and the number of noisy clients (N_K). Furthermore, the experiments regarding class balance and noisy clients were also run using the FedDF algorithm.

The experiments varying local epochs were repeated on the physical federation of Raspberry Pi's, both on Ethernet and Wi-Fi. This was chosen because altering this parameter changes the runtime of each communication round, while the behaviour for other tested parameters was approximately the same whether using the number of communication rounds or wall time as the x-axis.

All experiments used the baseline listed in Table 1 except for the parameter being varied in each experiment. These were chosen based on existing literature, in particular, [2, 3], and limited pilot experimentation.

K	S	α	E	L	N_K	B	η	γ
40	20	1	20	20	0	16	$5 \cdot 10^{-4}$	0.995

Table 1. Baseline parameters used for all experiments. Here, B refers to the training batch size.

3. RESULTS

Figure 2 shows the experiments run on the Pi setup. Note that these experiments are limited by time and have thus run different numbers of communication rounds. Table 2 shows the results of repeated experiments investigating different parameter choices after $L = 20$ communication rounds.

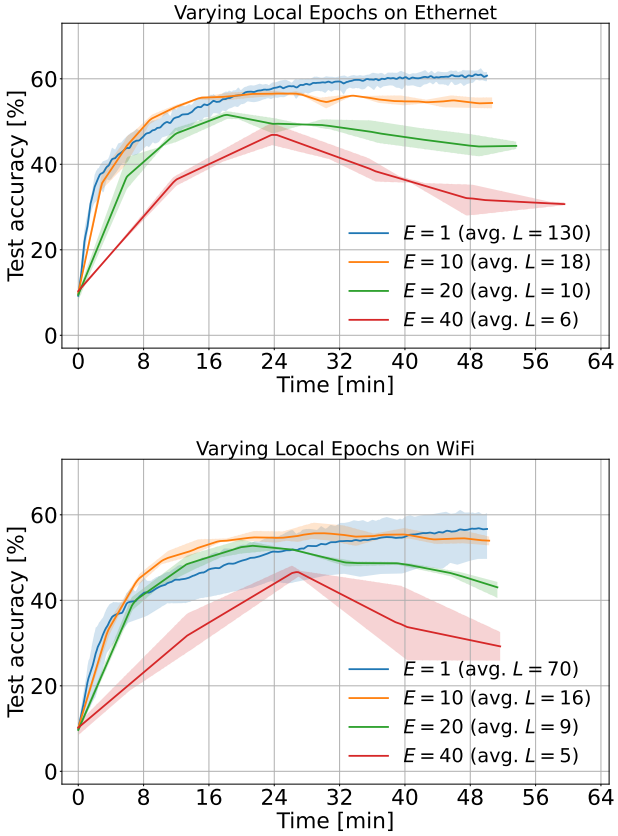

Fig. 2. Effect of the number of local epochs (E) using the Raspberry Pi setup on ethernet (top) and Wi-Fi (bottom). The lines show the mean accuracy of three repetitions, while the shaded areas outline the at any time best and worst of the repetitions. The legend also displays the number of rounds (L) each experiment completed before a timeout of 50 min.

Local epochs (E)			
1	10	20	40
48.0 ± 0.9	**52.2 ± 2.0**	37.6 ± 2.4	22.2 ± 2.0
Clients samped (S)			
5	10	20	40
35.4 ± 4.8	37.4 ± 2.6	**38.1 ± 2.0**	**38.1 ± 2.5**
Class balance (α)			
0.01	1.0	100.0	iid.
10.3 ± 0.6	36.7 ± 2.4	42.6 ± 3.4	**43.4 ± 2.5**
FedDF: Class balance (α)			
0.01	1.0	100.0	iid.
9.9 ± 0.2	55.8 ± 0.1	56.5 ± 0.9	**58.3 ± 1.0**
Noisy clients (N_K)			
0	10	20	30
37.1 ± 1.3	14.0 ± 2.1	10.3 ± 0.4	10.6 ± 0.8
FedDF: Noisy clients (N_K)			
0	10	20	30
54.5 ± 0.4	52.0 ± 2.9	51.6 ± 1.9	42.9 ± 0.3

Table 2. Final test accuracies [%] of FL models over $K = 40$ clients when running for fixed $L = 20$ communication rounds. Where nothing else is stated, parameters correspond to Table 1. Each run is repeated five times to produce an approximate 95% confidence interval.

4. DISCUSSION

Firstly, the results show that model convergence is possible even though data is distributed across devices; a conclusion substantiated by the fact that continuing the $E = 1$ training resulted in 95 % of centralised learning performance which is shown in Appendix Table 3.

From the Raspberry Pi experiments on Figure 2, we initially note that many accuracy curves fall during much of the learning. This effect is continually worse with more local epochs. We attribute this problematic behaviour to overfitting towards training sets, and, to check this explanation, visualise test performances during local epochs on Figure 3. Here, training accuracy on each client skyrockets during local epochs while the training accuracy plummets. The direct mitigation is to perform a structured hyperparameter search where better regularising measures for this task are tested. A more general idea is to perform local early stopping, but designing such a rule is non-trivial as we observe examples of the global, average model improving even though all local models overfit, as seen in early communication rounds and during the less biased learning for $E = 10$ shown on Appendix Figure 6.

When focusing on early learning less impacted by overfitting, the Raspberry Pi timing results reveal that it does not always hold that fewer local epochs are better. More local epochs can be optimal if training time is limited, as seen in minutes 5 to 15 using Ethernet and minutes 5 to 40 for Wi-Fi. During this time, the higher number of local epochs result in more time spent on training and less on communication, ex-

plaining why the this effect is most evident on relatively slow Wi-Fi. There thus exists a trade-off where E should not be too high to induce overfitting but not too low to slow down the rate of training data seen. When choosing this parameter, practitioners should be mindful of available training time and communication latencies in the system.

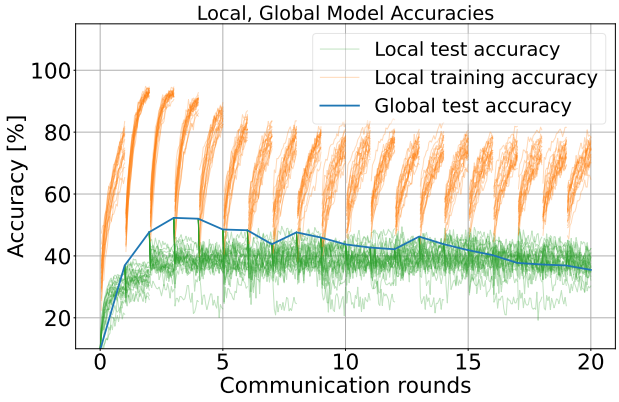

Fig. 3. A training trajectory when recording running accuracies on each client in each communication round where each faint line corresponds to $E = 20$ local epochs performances of each of the $S = 20$ sampled clients. For the $E = 10$ case, see Appendix Figure 6.

When running for a fixed number of rounds, as shown in Table 2, computational and communicational effects are disregarded, and $E = 10$ appears optimally as a compromise between under- and overfitting. For the number of clients sampled each round S, we note that even though datasets are somewhat imbalanced, $\alpha = 1$, FedAvg performs the same both when sampling half of the clients and when sampling all of the clients. Even using only five clients each round, the accuracy ended at 93 % of all 40, emphasising the stability of model averaging. This difference is largest in early rounds where lower values of S induce slower convergence, also visualised in Appendix Figure 7.

However, the class balance results show that this stability was removed when lowering α to 0.01. The learning improved significantly when training on more balanced classification tasks such as $\alpha = 100$ or iid.

For the FedDF experiments with the same hyperparameters, a generally much higher level of performance is noted. We speculate that the distillation procedure avoids the detrimental impact of the overfitted local models by performing a model fusion without averaging over big, bias-inducing parameters. This is backed up by Figure 4 where the problem of long-term falling performance is removed when using FedDF. Empirical prediction probability distributions might thus give a more robust characteristic of the learned knowledge than the model weights themselves.

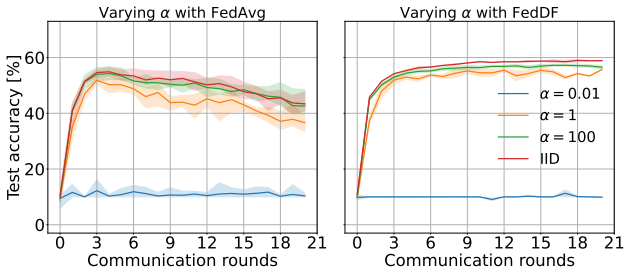

Fig. 4. Performance of the global model on the test set over communication rounds when varying data imbalance using both FedAvg (left) and FedDF (right). The lines indicate the mean performance of multiple runs, with the coloured areas outlining the least and most performant repetition at any run. The FedAvg experiments were repeated five times, with the much slower FedDF experiments only being run twice.

This added robustness is strongly exemplified for FedDF in the noise experiments. Here, having 10 out of the 40 data partitions being noisy ruins the performance of FedAvg thoroughly, while FedDF can perform reasonably even at 30 out of 40. Yet again, averaging probabilities instead of model weights appears to minimise the negative impact of fusing models with heterogeneous parameter values.

Though data label difference was simulated, use-cases such as smartphone computer vision or virtual keyboard language modeling have user data distributions that are much more diverse, e.g. one user only photographing their dog and another only landscapes, requiring analysis of data variety. In addition to these learning considerations, practitioners should be aware of the fact that FL is not the entire solution to user privacy as user data might be recoverable from trained models weights. To mitigate this, FL can be combined with differential privacy methods [9] to obtain maximally safe learning.

4.1. Conclusions

The case study performed in this project demonstrated that privacy-preserving learning on physical devices is possible as long as algorithmic choices are made with communication efficiency and robustness against data imbalance in mind. Using the simple, foundational FedAvg algorithm, running more local training on devices raised overfitting, but fewer epochs raised communication overhead. The distillation aggregation algorithm FedDF mitigated this overfitting and added robustness towards different data distributions at the cost of more computation on the central server. To investigate multiple realistic learning tasks, including computer vision models with higher performance, a federatio of devices with more computing power than Raspberry Pi's, such as smartphones, could be subject to the same analysis.

5. REFERENCES

[1] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao, "Advances and open problems in federated learning," *CoRR*, vol. abs/1912.04977, 2019.

[2] H. B. McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2017.

[3] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi, "Advances in neural information processing systems," 2020, vol. 33, pp. 2351–2363, Curran Associates, Inc.

[4] BEUC: The European Consumer Organisation, "Artificial intelligence: what consumers say," *BEUC Reports*, vol. Sep. 2020, 2020.

[5] Darrell M. West, "Brookings survey finds worries over ai impact on jobs and personal privacy, concern u.s. will fall behind china," *The Brookings Institution*, vol. Public Opinion Surveys on AI and Emerging Technologies, 2018, Visited Apr. 2022 https://www.brookings.edu/blog/techtank/2018/05/21/brookings-survey-finds-worries-over-ai-impact-on-jobs-and-personal-privacy-concern-u-s-will-fall-behind-china/.

[6] Alex Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[7] Ilya Loshchilov and Frank Hutter, "Sgdr: Stochastic gradient descent with warm restarts," 08 2016.

[8] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[9] Xingxing Xiong, Shubo Liu, Dan Li, Zhaohui Cai, Xiaoguang Niu, and Angel M. Del Rey, "A comprehensive survey on local differential privacy," *Sec. and Commun. Netw.*, vol. 2020, jan 2020.

A. MODEL

The model architecture is described below, listing the sequential operations in the forward pass.

Layer type	Hyperparameters
2D Convolution	1 in-channel, 16 out-channels, 3×3 kernel, stride of 1
ReLU activation	
2D Convolution	16 in-channels, 32 out-channels, 3×3 kernel, stride of 1
ReLU activation	
2D MaxPooling	2×2 kernel, stride of 2, no padding, dilation of 1
Dropout	$p = 25\%$
Flattening	
Linear w. bias	$6,272$ features in, 64 features out
ReLU	
Dropout	$p = 50\%$
Linear w. bias	64 features in, 10 features out

B. THE DIRICHLET DISTRIBUTION FOR DATA IMBALANCE

B.1. Impact of Dirichlet α

Se Figure 5 for illustration of varying degrees of data imbalance.

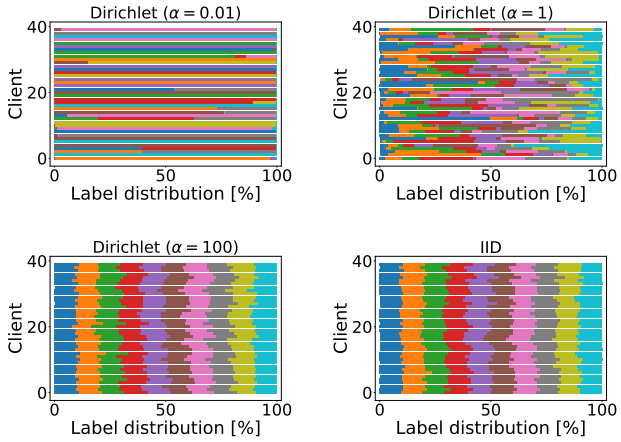

Fig. 5. Example 10-class label distributions over 50 clients for each of the tested sampling paradigms, where each colour corresponds to a label and each row to a client.

B.2. Sampling Algorithm

Let D denote a dataset of size $|D|$ with l different labels, of which there are $|D|/l$ each. The goal of the algorithm is to divide the dataset among C clients, such that the label distribution, $\boldsymbol{\pi}_i$ of each client i, follows the same Dirichlet distribution, $\mathrm{Dir}(\boldsymbol{\alpha})$, where every $\alpha_j \in \boldsymbol{\alpha}, j = 1 \dots l$ is the same value. For simplicity, the Dirichlet distribution will therefore be parametrized only by α; $\mathrm{Dir}(\alpha)$.

The distributions are structured into a matrix $\mathbf{P}$ of size $C \times l$, where the i'th row is $\boldsymbol{\pi}_i$. The ij'th element, P_{ij}, is then the fraction of label j on device i. Furthermore, the sum of the j'th column is the relative usage of label j scaled by the number of clients, C. As such, for every label to be used equally much, every column should sum to C/l. If any columns sum to more, the corresponding labels are oversampled, and so all of $\boldsymbol{P}$ needs to be normalized to make the largest column-sum equal C/l, causing some of the data to not be used.

A measure, u, is needed to determine how close $\mathbf{P}$ is to achieving the goal of making every column sum to the same. The lowest value of this measure should be achieved for a $\mathbf{P}$ where every column sums to C/l, while it should be progressively higher for poorer $\mathbf{P}$'s. We chose the L1 norm of difference in column sums and C/l:

$$u(\mathbf{P}) = \sum_{j=1}^{l} \left| \frac{C}{l} - \sum_{i=1}^{C} P_{ij} \right|$$

Similarly, standard deviation or the reciprocal of the entropy could be used.

The final thing to keep in mind before the algorithm is introduced is that reordering $\boldsymbol{\pi}_i$ makes it equally likely to be sampled from $\mathrm{Dir}(\alpha)$.

The algorithm changes the ordering within each $\boldsymbol{\pi}_i$ iteratively, until $u(\mathbf{P})$ is no longer lowered. The full algorithm is shown with pseudocode in Algorithm 1. The final $\mathbf{P}$ produced by the algorithm is normalized such that all data points with the most sampled labels are used exactly once.

In general, labels were found to be sampled more evenly for higher values of C and α. Running the algorithm for $C = 100$ and $\alpha = 0.01$ 100 times showed an average undersampling of less than 1 %, with the largest undersampling being 3.3 %. Only rarely was any one label undersampled by more than 5 %, a sign of the strength of the algorithm, even for a low value of α.

Algorithm 1 Dirichlet sampling algorithm

```
for i from 1 to C do                                   ▷ Initialize P
    P_i ← Dir(α)
end for
loop
    d ← Map()                                          ▷ Map swaps to u values
    for i from 1 to C do                               ▷ Do all possible swaps
        for all (j1, j2) ∈ (1...l) × (1...l), j1 ≠ j2 do
            Swap P_{ij1} and P_{ij2}
            d[(i, j1, j2)] ← u(P)
            Swap P_{ij1} and P_{ij2} back
        end for
    end for
    if min(d) < u(P) then
        i, j1, j2 ← argmin(d)
        Swap P_{ij1} and P_{ij2}
    else
        Break                                          ▷ No improvement possible
    end if
end loop
π ← Σ_{i=1}^{C} π_i
P ← P / max(π)                                         ▷ Normalize P by most sampled label
```

C. KL DIVERGENCE IN FEDDF

The KL divergence measures the distance from one distribution, Q, to a reference distribution, P. Let P and Q be discrete distributions defined on the probability space $\mathcal{X}$. The KL divergence is then defined as

$$D_{\mathrm{KL}}(P\|Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

A key property is $D_{\mathrm{KL}}(P\|Q) = 0 \Leftrightarrow P = Q$. However, the divergence is not a metric on the space of probability distributions, as in general $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$.

In the context of FedDF, the KL divergence uses the probability predictions of the student (or central) model as Q, while the target probabilities are used as the reference distribution P. The target probabilities are defined as the softmax of mean logits of the teacher models. The exact process is described in more detail by Lin et al. [3]

D. FURTHER RESULTS

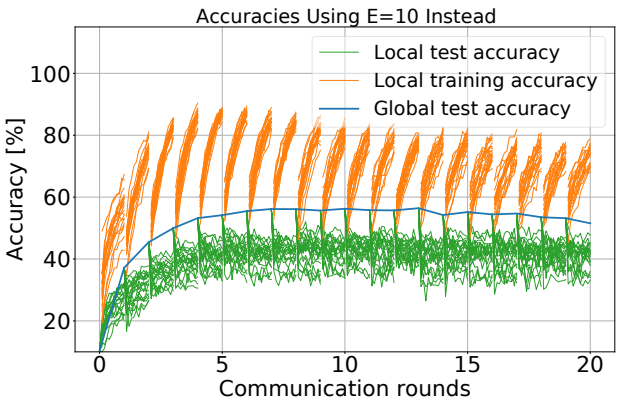

Fig. 6. A version of 3 with $E = 10$.

Algorithm	Steps before convergence	Final accuracy
FedAvg	200 comm. rounds	62.1 %
Centralised	10 full epochs	65.3 %

Table 3. Test set accuracy when running the above learning algorithms until test set accuracy stopped improving for 3 steps. The used FedAvg algorithm used $E = 1$ and otherwise baseline parameters presented in 1. The centralised learning algorithm used the full training each epoch, but otherwise same optimization approach as FL.

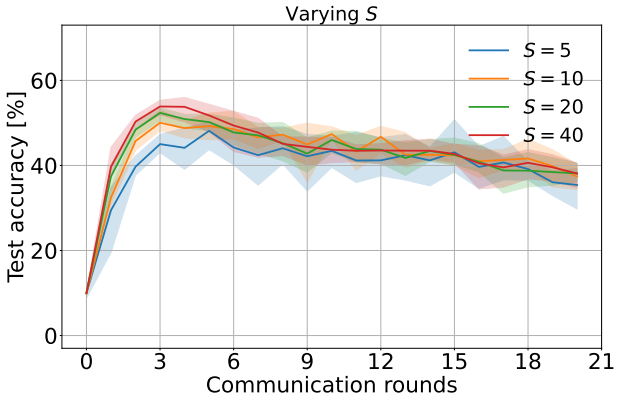

Fig. 7. Performance of the global model on the test at different communication rounds with varying number of clients sampled per round, S. The lines indicate the mean performance of five runs, with the coloured areas outlining the least and most performant repetition at any run.