Performance Evaluation 167 (2025) 102450

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

FedCust: Offloading hyperparameter customization for federated learning

Syed Zawad [a,1], Xiaolong Ma [a,1], Jun Yi [a,1], Cheng Li [b], Minjia Zhang [b], Lei Yang [a], Feng Yan [c,*], Yuxiong He [b]

[a] Department of Computer Science & Engineering, University of Nevada, Reno, Reno, 89557, NV, USA
[b] Microsoft, Bellevue, 98052, WA, USA
[c] Department of Computer Science, University of Houston, Houston, 77204, TX, USA

ARTICLE INFO

Keywords:
Federated learning
Hyperparameter optimization

ABSTRACT

Federated Learning (FL) is a new machine learning paradigm that enables training models collaboratively across clients without sharing private data. In FL, data is non-uniformly distributed among clients (i.e., data heterogeneity) and cannot be redistributed nor monitored like in conventional machine learning due to privacy constraints. Such data heterogeneity and privacy requirements bring new challenges for learning hyperparameter optimization as the training dynamics change across clients even within the same training round and they are difficult to be measured due to privacy. The state-of-the-art in hyperparameter customization can greatly improve FL model accuracy but also incur significant computing overheads and power consumption on client devices, and slowdown the training process. To address the prohibitively expensive cost challenge, we explore the possibility of offloading hyperparameter customization to servers. We propose *FedCust*, a framework that offloads expensive hyperparameter customization cost from the client devices to the central server without violating privacy constraints. Our key discovery is that it is not necessary to do hyperparameter customization for every client, and clients with similar data heterogeneity can use the same hyperparameters to achieve good training performance. We propose heterogeneity measurement metrics for clustering clients into groups such that clients within the same group share hyperparameters. *FedCust* uses the proxy data from initial model design to emulate different heterogeneity groups and perform hyperparameter customization on the server side without accessing client data nor information. To make the hyperparameter customization scalable, *FedCust* further employs a Bayesian-strengthened tuner to significantly accelerates the hyperparameter customization speed. Extensive evaluation demonstrates that *FedCust* achieves up to 7/2/4/4/6% better accuracy than the widely adopted one-size-fits-all approach on popular FL benchmarks FEMNIST, Shakespeare, Cifar100, Cifar10, and Fashion-MNIST respectively, while being scalable and reducing computation, memory, and energy consumption on the client devices, without compromising privacy constraints.

* Corresponding author.
E-mail address: fyan@unr.edu (F. Yan).
[1] Equal Contribution and Co-First Authors.

https://doi.org/10.1016/j.peva.2024.102450

Available online 16 November 2024

1. Introduction

Federated Learning (FL) has recently become a popular distributed learning paradigm that offers privacy and security protection while supporting collaborative learning across different data owners (a.k.a, clients) [1,2]. Unlike traditional distributed learning, in FL a shared global model is managed by a central server, and a random selection of client devices perform local learning with on-device data in each training round. The clients then send the local learning model parameters to the centralized server for aggregation (a.k.a, aggregator). Finally, the aggregated global model parameters are sent to a new random selection of the client devices to perform the next training round. In this process, since the user data never leaves clients, data privacy is preserved.

Although FL provides support on privacy and security [3,4], achieving high accuracy in a FL setting is challenging due to data heterogeneity [5,6], i.e., feature distribution is imbalanced among clients. Recent works try to alleviate the data heterogeneity impact by introducing their own methods [7–9], but data heterogeneity still remains an open challenge [10,11].

Similar to conventional training, learning hyperparameters (e.g., learning rate, batch size) play an important role in Federated Learning. Some recent works also explore hyperparameter customization for FL [12,13]. They adopt Random Search [14] or Reinforcement Learning [15] on each device to find the most suitable hyperparameters for the client. However, our experimental results show that the amount of resources required to perform such customized tuning is disproportionately high compared to the cost of local training. For example, the tuning algorithms require many hundreds of train-and-evaluate iterations while, in contrast, a client is typically selected only a few times to perform local training during the entire FL training process. This creates a vast discrepancy in the hyperparameter customization vs. actual training cost which brings the question of whether the benefits of hyperparameter customization is even worth it. In addition, the resources of client device in cross-device FL are usually highly constrained [16–18] and our results show that the amount of computational, memory, energy, and time costs required for tuning hyperparameters on most IoT and mobile clients may not even be feasible. There have also been ethical questions regarding the heavy usage of user-owned devices for training third-party models and thus reducing the cost on clients as much as possible is both urgent and critical [19,20]. Furthermore, existing works perform hyperparameter customization during each training round and thus can significantly slowdown the training speed, e.g., tuning a set of hyperparameters with 100 combinations is similar to perform 100x of local training and is thus roughly 100x slower training speed.

These observations lead us to a conundrum — performing Hyperparameter Customization on client devices is impractical due to the resource costs and the slowdown of training, but at the same time it is difficult to offload hyperparameter customization to the server without revealing client data or information. This motivates us to explore opportunities to perform hyperparameter customization without using user data or information. Through extensive characterization experiments, our key discovery is that it is not necessary to do hyperparameter customization for every client, and clients with similar data heterogeneity can use the same hyperparameters to achieve good training performance.

Based on the key discovery, we propose *FedCust*, a privacy preserving hyperparameter customization framework for FL that offloads the customization work to the server. In *FedCust*, clients are clustered into different heterogeneity groups according to heterogeneity measurement metrics such as "Heterogeneity Index" [7]. Clients in the same heterogeneity group use the same pre-tuned hyperparameters. To obtain pre-tuned hyperparameters, *FedCust* uses the proxy data from the initial model design to emulate different heterogeneity groups, i.e., create non-IID datasets with different heterogeneity metric values, and then perform hyperparameter customization for each group on the server side. In this way, the hyperparameters are tuned for each heterogeneity group without accessing any client data nor information. To make the hyperparameter customization scalable, *FedCust* further employs a Bayesian-strengthened tuner to significantly accelerates the hyperparameter customization speed. The pre-tuned hyperparameters are stored in the *Hyperparameter Reference Table* (HRT) - a table whose row and column combinations embody a specific heterogeneity group and the cell's value is its corresponding optimal hyperparameter set. The HRT is very lightweight (typically less than 1 KB) and is distributed to clients with model parameters. Clients choose the hyperparameters based on their heterogeneity group.

We prototype *FedCust* on a real distributed FL testbed and evaluate its effectiveness and robustness using popular and sophisticated FL benchmarks. Our evaluation results show *FedCust* offers superior performance over existing methods that are heterogeneity-oblivious (such as [1,7,21,22]) and are on-par with the significantly more expensive on-client tuning methods [12,13]. For FEMNIST of LEAF [23], the most popular FL benchmark, we achieve up to 7% accuracy improvement as well as 2/4/4/6% better accuracy than the widely adopted one-size-fits-all method for the Shakespeare, Cifar100, Cifar10, and Fashion-MNIST datasets respectively. We show empirically that our method is scalable with the number of devices in the system, generalizable to different data heterogeneity definitions and granularity, and completely reduces client-side tuning resource costs.

In summary, *FedCust* has two major advantages over all prior art. First, it completely offloads all tuning costs to the server and imposes no additional overhead on resource-constrained client devices, which makes hyperparameter customization practical for cross-device FL systems. And second, no client data is exposed throughout the whole process, which allows for preserving of privacy constraints.

2. Background and related work

2.1. Data heterogeneity in federated learning

Data heterogeneity is an essential property of FL as clients usually have different amounts and distributions of data [24,25], with the extent of data heterogeneity being much more pronounced than in conventional ML [10,25]. As such, there are lots of

recent works targeting at addressing this issue. The first line of works focus on designing a client selection policy for choosing devices with similar data distribution, such as [9,26–29]. Another line of works propose strategies to identify which parts of the model are mostly affected by data heterogeneity, and then apply regularization methods (such as weight regularization) to correct them, some examples being [19,30]. Finally, another line of works innovate on the aggregation algorithms to mitigate the data heterogeneity impact in FL. For example, [31] proposes three different server-side optimization algorithms. FedDF [32] allows flexible aggregation over heterogeneous client models by using ensemble distillation, while [33] uses means and medians instead of mean for aggregation. Some papers such as [34–36] also fundamentally change the system architectures by enabling server-side training and hierarchical aggregation. In this work, we focus on hyperparameter customization, which is complementary to all the above works and we demonstrate in our evaluation that together with our methodology, the performance can be further improved.

2.2. Hyperparameter optimization

2.2.1. One-size-fits-all hyperparameter optimization for FL

The state-of-the-practice hyperparameter optimization in FL is to hand tune a single set of hyperparameters and apply them to all clients indistinguishably. Some recent works aim at improving this process by using different approaches. [37] adopts Bayesian Optimization to tune the local training hyperparameters on-device individually, but does not customize the hyperparameters individually. Rather, it trains the acquisition function based on clients' data and uses it to predict a final global hyperparameter set for the full system. [38] performs local Bayesian Optimization to find the best hyperparameters for differential privacy parameters, but not for local training hyperparameters. [39] performs an evaluation of the [37] in an industrial setting, but does not propose anything new. In addition, they perform the hyperparameter optimization on client devices, which slows down the training process and degrades user experiences due to significantly longer computation and higher power consumption. In comparison, *FedCust* offloads the tuning overheads to the server. FLoRA [40] optimizes hyperparameters by formulating a non-linear loss function and optimizes it for decisions trees, but the approach is not generalizable for deep learning models.

2.2.2. Hyperparameter customization for FL

There are a few works explore hyperparameter customization for FL, but they either focus on certain aspect or require information sharing that violates FL privacy requirements. *FedProx* [7] and *FedNova* [41] hand tunes the local number of SGD steps to reduce the communication overheads, but such an approach does not apply to general hyperparameters. *FedTune* [42] focuses on tuning systems hyperparameters instead of learning hyperparameters to minimize resource usage and their approach also requires expensive local tuning. Genetic CFL [43] observes data directly to assign similar clients together into clusters. [13] uses Reinforcement Learning (RL) to adapt the learning rates of clients over training rounds. It requires client information to train the RL model and the tuning is performed on client devices, which slows down the training process and degrades user experiences due to significantly longer computation and higher power consumption. In comparison, *FedCust* is a more general approach that supports other hyperparameters beyond learning rate, batch size and offloads the tuning overheads to the server to reduce the burdens of client devices. *FedCust* is designed to support a comprehensive range of commonly used parameters without restriction. The parameters selected for demonstration in our paper were chosen specifically to showcase the effectiveness of our design. This holistic approach ensures that all aspects of the model training process can be optimized to enhance overall performance. *FedCust* employs Bayesian Optimization which is much more lightweight than RL, and requires no client information to meet stronger privacy requirements. *FedEx* [12] uses One-shot Neural Architecture Search to develop customized models for each client, and also tunes the learning hyperparameters along this process. Because they perform both architecture search and hyperparameter optimization on client during the training, the overhead is even more expensive. We compare *FedCust* with [12,13] in our evaluations to further validate the advantage of *FedCust*.

3. Federated learning hyperparameter optimization study

In this section, we first present a primer of FL. Then we explain why heterogeneity-aware hyperparameter optimization is critical in Federated Learning. Finally, we discuss why it is prohibitively expensive to perform client-side hyperparameter customization and whether it is possible to address this challenge by offloading hyperparameter customization to the server side.

3.1. Federated learning: A primer

Problem Formulation. The federated learning problem aims at learning a single, global model from data stored distributedly on a large number of remote devices. Different from conventional distributed training, FL has the constraints that device generated data is stored and processed locally, with only intermediate model updates being communicated periodically to a centralized server. The server then performs an aggregation [44]. More formally, the goal of FL is to minimize the following objective function:

$$\min_\theta F(\theta), where\ F(\theta) := \sum_{i=1}^{m} p_i F_i(\theta, D_i) \tag{1}$$

Here, m is the number of devices. p_i is a coefficient that determines the impact of each device, usually set to 1/m. $F_i(\cdot)$ is the local objective function for the ith device, and D_i represents the local data on the ith device.

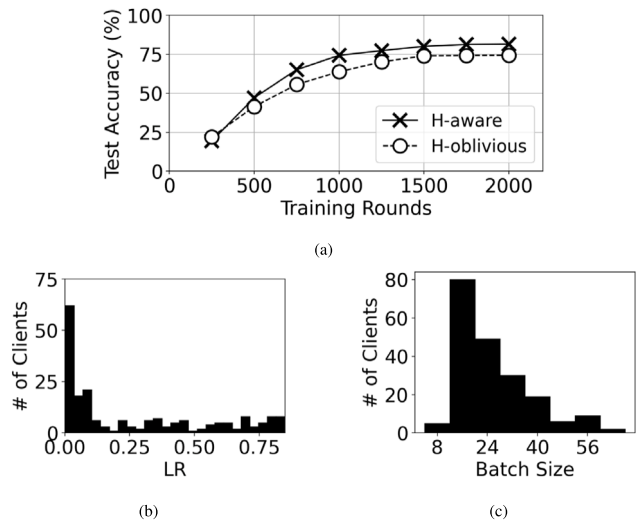


Fig. 1. Heterogeneity Impact - (a) Test accuracy vs. training rounds comparison. (b) The tuned learning rates and the corresponding number of clients that use them. Derived after hand-tuning all clients. (c) Scalability of the number of training steps that must be run to tune hyperparameters with varying number of clients in the system.

Data heterogeneity. One big challenge in solving the above problem lies in the heterogeneity in data D_i. In FL, each client has its own private data with different *data distribution* and *data quantity*. This data heterogeneity adds complexities in solving Eq. (1), because it violates the independent and identically distributed (i.i.d.) data distribution assumption that many statistical optimization methods (e.g. Stochastic Gradient Descent) rely on [25]. To further understand how data heterogeneity affects FL performance, we need to look deeper into where data heterogeneity comes from. At the high level, there are two types of data heterogeneity: heterogeneity in data distribution, and heterogeneity in data quantity. *Data quality heterogeneity* means the data from different clients may have different types and features, which are often associated with the user behavior of the client devices. Take image classification of cats and dogs as an example, cat-owners usually have more cat images than dog images on their phones. Such data distribution heterogeneity may cause performance issues, e.g., a model trained on cat owners may have better performance on cat images than dog images, and vice versa [1,7,25,29]. Some prior works propose different metrics to quantify heterogeneity in data distribution for analysis and sensitivity experiments. For example, [45] uses Poisson distribution to synthetically distribute the datasets across clients, while [46] uses Gaussian distributions. Recent papers use real-world distributions such as LEAF [23]. However, how to quantitatively measure heterogeneity in data distribution remains a challenging and open question.

Data quantity heterogeneity means that the amount of data may vary from client to client. This is also due to user behaviors. For example, clients who text a lot have more data points to train for a word-prediction model than clients who text very little. Such heterogeneity also impacts performance during the training process.

Privacy constraints. The problem gets even more complicated when taking privacy constraints into account, where the private data of clients cannot be monitored nor manipulated (e.g., to create more balanced data across different devices). This means that even if we can measure and characterize the data heterogeneity of a client, that data heterogeneity information must remain as a black-box and cannot be shared outside of the client device. Without knowing the data heterogeneity information, conventional wisdom for mitigating data heterogeneity impact is difficult to be adopted in FL [5], making it even harder to improve the FL process based on data heterogeneity.

3.2. Heterogeneity-oblivious vs. Heterogeneity-aware hyperparameter optimization

In this section, we conduct an empirical study to understand the importance of heterogeneity-aware hyperparameter customization, which adaptively chooses hyperparameters for individual client devices in FL. For detailed experimental setup, please refer to Section 6.1. Our key intuition is that given the heterogeneity in client data, the hyperparameters for each client device should be customized based on the data heterogeneity to facilitate the local learning process, and by improving the local learning of each individual client we can obtain an overall improved global model. To test our hypothesis, we design a simulation experiment to compare the performance of the following two different approaches.

- ***Heterogeneity-oblivious (H-oblivious)***: Following the common practice of FL (such as FedAvg [1]), we hand tune a global set of hyperparameters and use it across all clients.
- ***Heterogeneity-aware (H-aware)***: We hand tune the hyperparameters for each client to have customized hyperparameters per client.

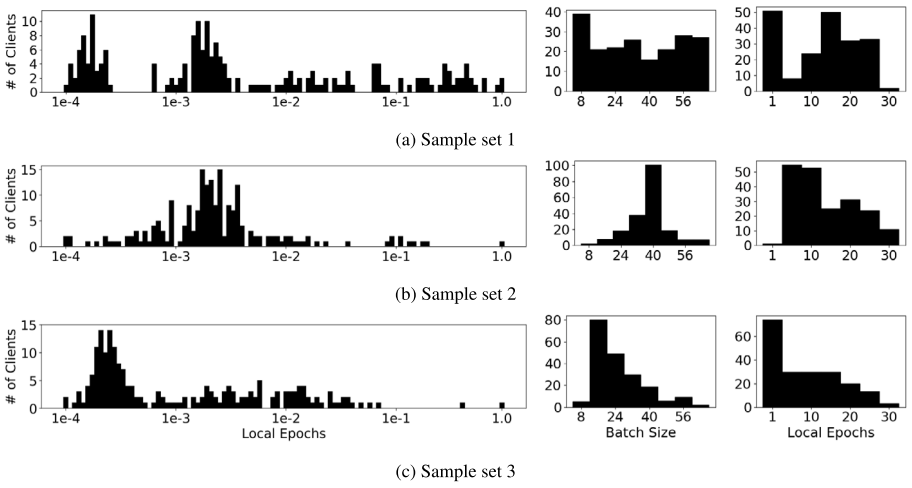


Fig. 2. Heterogeneity-aware Tuning Configurations - The Learning Rate, Batch Size and Local Epoch parameter value distributions across distinct client sets chosen after Heterogeneity-aware tuning. Each of the client sets is mutually exclusive and sampled as 10% from the full LEAF dataset.

In the experiment, we use FEMNIST from LEAF [23], a popular image classification benchmark with data heterogeneity (see Section 6). We hand tune the local hyperparameters learning rate, batch size, and local epochs (defined as the number of training epochs at each client) for each client. We follow the literature [23] to set the search range of hyperparameters for the learning rate between 0.0001 to 0.1, the batch size between 8 to 64, and the local epochs between 1 to 30 which gives us a total number of combinations (or search space) of 24,000 (details explained in the next section). We apply grid search to identify the best hyperparameters for individual clients. The distribution of the learning rates are given in Fig. 1(b) (batch sizes are given in Table A.6).

The comparison of test accuracy curves across training rounds is shown in Fig. 1(a). We can see *Heterogeneity-aware* outperforms *Heterogeneity-oblivious* in accuracy during the training process and yields a 7.4% better accuracy after the accuracy plateau. This is presumably because the one-size-fits-all approach of *Heterogeneity-oblivious* is inevitably unfavorable for some clients no matter how judiciously the tuning is due to the data heterogeneity across clients. On the other hand, the customization approach of *Heterogeneity-aware* can tailor the hyperparameters for each client to maximize the performance benefits. Fig. 1(b) shows the hand-tuned learning rate distribution across clients. The distribution demonstrates that while a majority of the learning rates are within the $0.0001 - 0.1$ range, there are lots of clients requiring a much more diverse range of values, making it difficult to choose one set of hyperparameters that work well for all clients. These results indicate that *data heterogeneity-aware hyperparameter optimization approaches have the potential to improve the learning performance compared to data heterogeneity-oblivious approaches.*

To further understand the impact of data distribution on the impact on the hyperparameter choice, we perform experiments where we vary the underlying data heterogeneity of the clients and derive their *Heterogeneity-aware* hyperparameters and observe their differences. In Fig. 2, we sample different client distributions from LEAF's FEMNIST dataset. It contains 62 classes of 805K black-and-white 64×64 images which are hand-written by 3550 different users, and each user is represented as an individual client. Thus the dataset provides a natural data distribution by default compared to other standard datasets which are usually IID by nature and needs to be artificially split. Each of the clients also contain a variable amount of datapoints, thus representing data quantity heterogeneity as well. For all our experiments, we randomly sample 200 clients (as is the standard practice [7,8,47]). For this experiment, we sample different parts of the full dataset.

In Fig. 2, we show the results of the *Hyperparameter-aware* hyperparameter choices for different dataset samples. The samples are chosen such that only clients with certain data quantities are used as the 200 clients in the full FL system. Lower data quantity clients contain less number of classes and vice-versa, making it a simple metric for deriving different data heterogeneity distributions. Sample sets 2 and 3 contain clients with >400 and <150 datapoints respectively while set 1 has a mixture of both. From the differences in distribution, we can clearly see that the type of sampling has a significant impact on the hyperparameter choice. For Sample set 3 we have low number of datapoints and so seems to favor low learning-rates, batch sizes and local epochs since they tend to be better at reaching the global minima faster for low-noise planes [48] (while the full system is heterogeneous, the data within each client tend to be similar and thus less noise in the local datasets). For Sample set 2, we see the opposite effect due to more datapoints and classes per client. Lower learning rates and batch sizes would result in overfitting on the larger and diverse datasets and so are avoided. For Sample set 1, we see the hyperparameters are relatively more evenly spread out since it is a mixture of both. These observations indicate that the underlying data quality and quantity distributions influence the tuning results significantly, and there is a relationship between heterogeneity and chosen hyperparameter sets which we can exploit for reducing the search space.

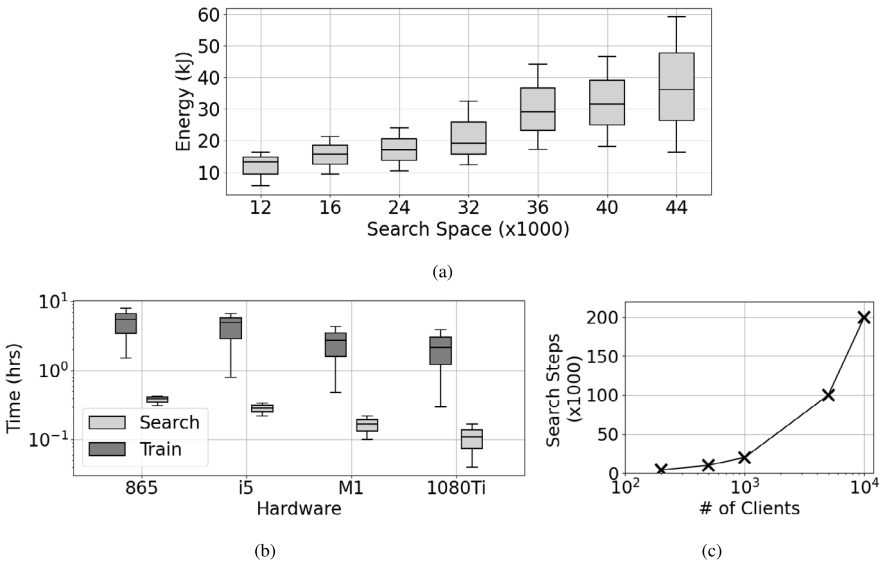


Fig. 3. Resource Cost and Scalability - (a) Energy spent per client for tuning hyperparameters (b) Time taken for each client for tuning and FL local training (c) Scalability of the number of search steps that must be run to tune hyperparameters with varying number of clients in the system. The energy and time is measured using the Android Profiler [49] over 200 clients with Samsung S20 devices.

3.3. Resource cost and scalability

Heterogeneity-oblivious methods only tune one set of hyperparameters and use it across all clients, so the tuning cost is not associated with the number of devices. However, given that there could be millions or even billions of remote devices, data heterogeneity-aware methods that perform customized tuning on each client would incur significant accumulated cost. More importantly, clients in FL are usually IoT or mobile devices with limited computing capacity. Performing computing intensive hyperparameter tuning tasks on these devices is slow and power demanding, which may significantly discourage user participation. For this section, we evaluate the resource cost of *Heterogeneity-aware* tuning on low-powered devices to understand the cost-benefit tradeoffs of performance benefit vs. resource usage.

For the first experiment, we compare the total power consumption on each client in the tuning phase. Fig. 3(a) shows the median and quartiles of energy consumption of all 200 clients against the search space. Here we see that the energy consumption is significantly high during the hyperparameter tuning phase. The most common mobile devices tend to have between 60kJ to 240kJ of total battery life [50,51], and given that they usually run with multiple processes in the background, this load can make it infeasible to tune on-device. Additionally, works such as [8,17,52,53] make the argument that the FL process must be minimally invasive which may not be possible here as well. Next we look at the time spent tuning vs. training on-device across a wide range of hardware (the results are presented in Fig. 3(b)). Here we train for 2000 rounds for 200 clients with 20 clients selected per round. For tuning, we use the search space described above. The probability of having a client participate is very low. Even with a high number of epochs, the small number of times a client gets selected means very minimal time is spent on the local training process, which is a desired system property. Time spent tuning, however, requires many more training runs and is therefore exponentially more expensive than the actual training phase. In resource-constrained systems, the clients' may not be willing to bear this extra cost for the boost in final model performance. These two results shows a significantly more cost in both time and energy for Heterogeneous-aware approach since tuning hyperparameters requires considerable trial and errors, As pointed out in [21,29,53], clients can only be selected under certain circumstances, such as when the devices are plugged in, not being used, with sufficient memory, to avoid impacts on the user experience. Under such criteria, a significantly longer tuning and training time, resource consumption may prevent a large portion of clients from participating in the training process.

Scalability is also a challenge for heterogeneity-aware tuning approaches. For the heterogeneity-oblivious tuning approach (the global tuning), the traditional method is to train the full FL system for a few rounds for each hyperparameter set being trialed [27,29]. For the *Heterogeneity-aware* approach, we explore each hyperparameter set by first selecting a subset of clients randomly and training the client with the set for a few local epochs and evaluate its final model performance, eventually selecting the best set for that client. In Fig. 3(c), we show the total number of training iterations involved to find a good set of hyperparameters (i.e. Search Steps) against the number of clients involved in the training process. We see that there is a linear increase of the number of search steps proportional to the number of clients. given that the tuning cost is already expensive for each device, this linear scaling is also highly costly for the full system. As such, in order to design a resource-efficient *Heterogeneity-aware* FL tuning framework, we must offload the search phase from the clients.

Table 1
Optimal learning rates - Under different data heterogeneity levels, batch sizes, and data sizes.

Heterogeneity index	Batch size/Number of data points			
	5/400	10/400	20/800	30/800
0.15	0.021	0.04	0.041	0.061
0.50	0.032	0.065	0.071	0.105
0.75	0.042	0.086	0.081	0.125

4. Hyperparameter customization offloading

Considering the prohibitively expensive cost for client-side *heterogeneity aware* hyperparameter customization, one natural question is: *would it be possible to offload hyperparameter customization to the server side to reduce the cost?* Even though offloading the hyperparameter customization process to the server side seems straightforward as servers are usually hosted in the cloud or data centers with plenty of computing resources, it is actually very challenging. Due to the privacy constraints in FL, neither the data nor the properties of data (e.g., including the data heterogeneity information) of a client can be shared with the server. Without knowing the data properties, offloading tuning to the server becomes difficult as the hyperparameters are blind to the clients' needs.

Given the restrictions on client data, directly customizing hyperparameters using client data on the server side is infeasible. However, given that hyperparameters are highly influenced by data distribution, it raises an interesting question: can we categorize datasets based on their data distribution characteristics and apply the same set of hyperparameters to each category? In other words, can hyperparameters that are tuned to account for data heterogeneity provide consistent results across various datasets? To test this out, we first have to define a quantitative measure of data heterogeneity. We define *Heterogeneity Index* following literature [25,29,54]. *Heterogeneity Index*, denoted as $HI(c)$, is defined as a normalized measurement of data distribution heterogeneity:

$$HI(c) = 1 - \frac{1}{c_{max} - 1} \times (c - 1), c \in [1, c_{max}] \tag{2}$$

where c controls the heterogeneity by adjusting the number of classes per client out of the total number of classes c_{max} in the full dataset. $HI(c)$ ranges from 0 to 1, where 0 represents a completely balanced synthetic dataset and 1 means there are only data points with 1 class on the device, which is the highest level of imbalanced data distribution possible. We then run a simple experiment where we use 6 different datasets with data quantity of 400 and 800 combined with *HI* of 0.15, 0.5, and 0.75 respectively. We set the batch size of 5 and 10 for 400 samples dataset, and 20 and 30 for 800 samples dataset respectively. We hand tune the learning rate to achieve the best performance. The optimal learning rates is shown in Table 1. We can see the optimal learning rates have a clear pattern – *with the increasing of heterogeneity level while other factors are the same, the learning rate increases.* Also, with more training steps (the ratio of data set size and batch size), the learning rate also increases. This observation is corroborated by the paper [48], where they derive the relationship between the noise scale, i.e., the magnitude of the random fluctuations in the training dynamics, and the learning hyperparameters. Specifically, they suggest that the learning rate should increase with increased noise scale. In the federated learning case, the higher the heterogeneity, the noisier the training process [25,55] which is why we observe that an increase in data heterogeneity requires higher learning rates. Such a pattern seems to be helpful in making offloading hyperparameter customization on the server side possible even without the client data, since we can estimate the learning rate for a client device based on the shared pattern in data heterogeneity. Fig. 4(a) shows the pattern of the optimal learning rates for the system's corresponding *HI* (Hand-tuned LR). We fit a quadratic regression model (Estimated LR) and interpolate the LR values for other *HI*. Fig. 4(b) shows the difference of the accuracies derived with the interpolated learning rates against their optimal hand-tuned values. For example, here if we estimate the learning rate at *HI* 0.36 via interpolation, the estimated learning rate is 0.029, very close to the actual optimized learning rate 0.034. However, we observe from Fig. 4(b) that the accuracy when using the estimated learning rate is around 15% lower than the accuracy of tuned learning rate. This study suggests it is challenging to utilize the patterns of hyperparameters for estimating optimal hyperparameter values to reduce the tuning cost since the learning rates are very sensitive. However, this pattern is sufficient such that it can be used by the acquisition functions in BO as a guide during the exploration phase e.g., by avoiding exploring known bad hyperparameters (see Table 2).

5. *Fedcust* : Heterogeneity-aware hyperparameter optimization

In summary, we make the following key observations from the above sections. First, there can be significant increase in final model performance if hyperparameters are tuned to fit the local data than using a single global *Heterogeneity-oblivious* set, and that the data distribution per client is influential here. Second, privacy requirements means that we have to tune hyperparameters on the client hardware if we are to achieve good results. Third, local tuning is prohibitively expensive, especially on mobile hardware which is the majority of the system in cross-device FL and so we must design a technique to reduce this burden. Lastly, it is possible to offload hyperparameter tuning to the server, but special considerations must be taken into account when doing so. Based on these insights grained, we propose an efficient and scalable hyperparameter customization framework named *FedCust* that offloads hyperparameter customization to the central server without violating privacy constraints. We describe the proposed system in this section.

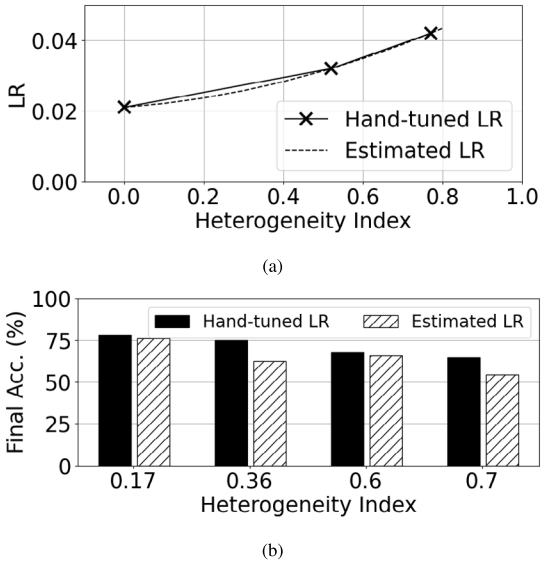


Fig. 4. Comparison of Tuning Methods - (a) Hand-tuned vs. estimated Learning Rate (LR) under different heterogeneity index. Estimation is done via regression fitting. (b) Final test accuracy comparison between hand-tuned vs. estimated LR under different heterogeneity index.

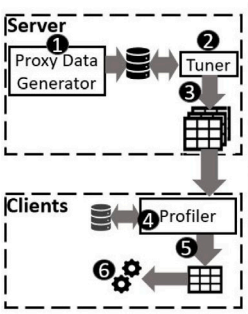
Table 2
Training setup - Describes the model, number of train/test datapoints, clients and global hyperparameter sets.

Dataset	Model	Train/Test split	Clients Total/Per round	Global LR/Batch size	Training rounds
FEMNIST	2 conv 2 dense	49,644/6,200	192/10	0.0004/8	2000
Shakespeare	128 hidden size LSTM	74,000/13,000	10/1	0.0003/4	100
Cifar100	Resnet18	50,000/10,000	50/5	0.045/16	1000
Cifar10	4 conv 2 dense	50,000/10,000	50/5	0.05/16	500
F-MNIST	2 conv 2 dense	50,000/10,000	50/5	0.002/8	500

5.1. Proxy dataset-based hyperparameter customization

Based on our observations, we know that per-client customized tuning based on data distribution can yield a significant boost in model performance, but privacy and resource constraints can make this impossible for cross-device FL systems. The main problem here is to enable hyperparameter tuning to be performed on the server without being able to directly access client data. We tackle this problem by using a representative dataset called the *proxy* dataset. The main idea for *FedCust* is to manually configure this dataset to be representative of the underlying clients' data distributions and perform the hyperparameter customization process on it on the server side. This allows us to search hyperparameters *without any client information nor accumulated client information being shared with the server.* and *offloading the tuning overhead to the server completely*. Our observations from the previous section showed that the hyperparameters are correlated with data heterogeneity. Using this intuition, we can create a set of proxy datasets with different heterogeneity characteristics and tune the hyperparameters on those sets. Then by grouping clients based on their data heterogeneity characteristics to a similar proxy set, we can reuse the corresponding tuned hyperparameters of the proxy set on the clients.

Now, the main question here is how to derive a good proxy dataset such that it is representative of the underlying data. In practical FL settings, proxy datasets are quite common. They are used for developing the model architecture and tune the global hyperparameter set. In practice, such a dataset can be provided by the model developers, or from user-shared/publicly available data [56,57]. For example, the initial training datasets used by the model developers when designing the architecture are derived from public datasets [11,58], or some datapoints are scrapped from consenting users to tune the global hyperparameters [8,56,57]. We can use the same datasets for our case as well for two main reasons - (1) the model itself is designed based on this proxy dataset so they are well suited to each other, and (2) the global hyperparameter is tuned on it to capture the correct features in the first place. Therefore, the proxy dataset only needs to reflect some general information about the task such as the number of classes and input dimensions. This is done in practice by using similar datasets. For example, MNIST [59] is used as proxy for Fashion-MNIST [60] due to having similar input features and classes even though they are separate datasets. The important point to note here is that this



❶	Proxy Data Generator	Generate proxy dataset with different quantity and distribution heterogeneity.
❷	Tuner	Hyperparameter tuning on generated proxy dataset using Bayesian Strengthened Tuner.
❸	Populate HRT	Populate Hyperparameter Reference Table (HRT) and send to each client.
❹	Client-side Profiler	Profile data quantity and distribution heterogeneity of the client.
❺	Look up HRT	Use profiled information to look up the HRT and apply the corresponding hyperparameters for training.
❻	Local Training Module	Perform local training using selected hyperparameters.

Fig. 5. *FedCust* System Design - Shows the major steps involved in the tuning process.

allows us to have a representative dataset on the server without violating the clients' privacy. We do an analysis on the impact of proxy dataset choice in the Evaluation section to better understand how it impacts the hyperparameter search and training phases.

5.2. Privacy-preserving hyperparameter customization via hyperparameter reference table

Since the server does not have the client data, it is challenging to figure out what exact data heterogeneity the client data may exhibit. To resolve this issue, *FedCust* takes a reference table based approach, where *FedCust* would explore a large number of data qualities and quantities that represent different combinations of data heterogeneity. For each data heterogeneity point, *FedCust* would let the server to perform a hyperparameter tuning to identify a set of promising hyperparameters under that point. The results are recorded in a *Hyperparamter Reference Table* (HRT). HRT is a two-dimensional array where the rows are the data distribution heterogeneity (e.g. Heterogeneity Index) and the columns are the data quantity. Each cell contains the hyperparameter sets for its combination of quality and quantity properties.

In order for this HRT to function, we need to quantify the data quality and quantity such that each cell can represent a certain type of data distribution as a measure of its combination of quality/quantity values. Quantity is a direct metric, but assessing data quality is non-trivial. Ideally, data quality should be a measure of how much useful feature representations can be learned by a model from that dataset. While this is well understood conceptually in literature [7,25], there is yet a formal quantifiable definition for it, and this problem is over-arching for ML research in general. Data heterogeneity is a commonly-faced issue in FL and therefore many works have provided metrics to quantify it in their own way. However, there has yet to be a single formal definition and so we adapt these metrics for our purposes. These measures are usually based on class-wise random sampling methods. Some of the most common definitions use Gaussian [61,62], Poisson [63,64], Dirichlet [65,66] distributions or the HI value for sampling the from the full dataset. For example, the Gaussian mean and variance parameters determine the spread of the sampling of the number of classes where higher variance values mean more evenly distributed the classes. Poisson and Dirichlet can also function similarly, and HI functions as explained above. They are convenient to use since these functions are controllable, well-understood and generalizable to different datasets (for non-classification tasks such as next-word prediction [67] we can sample based on the similarity of the output word vectors [68]). While not directly a measure of "quality", they have been found sufficient by the latest FL literature and so we use them for our purposes as well.

After setting up the HRT with our chosen metrics, we sample from the proxy dataset such that the cell metrics (i.e. data quantity and quality) are met. We then perform hyperparameter search to find the best hyperparameter set for that cell on that sampled proxy dataset subset. We explain the search process in the next sub-sections. Once the HRT (example given in Table A.7) is fully populated in this way, it is sent *one way from server to clients*. The table is typically a few KBs and sent with the global model, thus the networking overhead is negligible. On the client-side, *FedCust* has a profiler that measures the local dataset's data quality and data quantity. This is profiler uses suitable distance metrics based on the quality metric used. For example, for HI, the *Euclidean* distance is used. The client's data heterogeneity is first calculated *on the client side* by measuring the number of classes and datapoints, which can give us the HI number. Then the *Euclidean* distance between each of the cell's row/column HI/quantity and the current client's measured HI and quantity is taken, and the most similar cell type to the client's distribution is determined to be the one with the lowest distance value. The client then uses the chosen cell's hyperparameters for local training. Similarly, for Gaussian, Poisson and Dirichlet distributions we use the Chi-squared [69] or B-distance [70] values. Note that our framework is generalizable to any type of such metrics, even novel ones that do not rely directly on class quantities such as HI. For the rest of our paper we use HI as an example since it is most intuitive but it should work well regardless of the choice of metric. The main idea here is to be able to derive an HRT that can sufficiently capture the possible clients' heterogeneity, which can be done with any suitable distribution and distance metrics. We provide empirical results using different metrics in Section 6.

It is important here to note here that while the generation of the HRT occurs at the server, the distance between the cells and the individual client's local data distribution measures are only kept at the client. Only the client knows the combination of quantity/quality that it has and selects the hyperparameter by itself. The server acquires no information from the clients at this stage, making this a fully private mechanism.

By having each client look up the HRT to choose the best matching entry based on its profiler's HI measure, the client then can use the customized hyperparameters provided by HRT for local training. The rest of the FL training proceeds as usual. A description of all the steps and a complete system overview is given in Fig. 5. Note that in this process, *FedCust* does not collect or monitor client data, so it *respects the privacy constraints* of FL. At the same time, it also imposes almost *no additional overhead* to the clients.

5.3. Determining HRT granularity

Since the hyperparameters are tuned across each cell of the HRT, it is a direct determinant of the search space along in addition to the hyperparameter ranges. The higher the granularity (i.e. the number of quantity/quality cell combinations) the more the search overhead but more fine-grained the tuning and thus better local training results. Therefore, we need a mechanism to determine the best suited granularity for the HRT. We do this by first setting up and populating a low-granularity HRT (for example, with 24 total cell combinations as in Table A.7), then selecting a client and measuring its data distribution distance to the closest HRT cell as described above. Note that this too is generalizable to the heterogeneity metric choice. If this distance value is above a certain threshold τ, the client sends a signal to the server to ask it to increase its granularity. The server then increases the number of quantity/quality rows and columns and finds the new hyperparameters for the new combinations, and FL training continues as usual. The τ value functions as a tradeoff threshold between search cost and tuning performance. We analyze its effect in Section 6 in more detail. This system is also private since the server still has no idea about the client's data distributions but only that the current available cells are insufficient for tuning.

At implementation, this calculation of the HRT granularity can be performed both before the training begins and online. The former method can be applied by selecting client in a round-robin fashion to determine whether they require finer granularity for HRT. The latter method can be performed during training when a client gets selected. Either way is equally effective, but the latter method may cause delays per round if in the worst case scenario every client ends up asking for finer granularity. For the experiments in Section 6, we first try our basic *FedCust* implementation with a static HRT to demonstrate that it is an efficient FL hyperparameter tuning platform. We then add the HRT granularity determination on top of it to further enhance it for use in real-world scenarios.

5.4. Scalable hyperparameter customization via Bayesian strengthened tuner

One challenge raised from creating the HRT is that because we have to cover a wide range of data heterogeneity (i.e. data quantity and distribution), as well as different hyperparameters (e.g., learning rates, batch sizes), the combinations can be extremely large and the customization process can still take excessively long to run even on the server side. To accelerate the customization speed, we use **Bayesian Optimization (BO)** as our tuning method as it has been proven to be quite useful in hyperparameter optimizations [71,72]. Here we leverage BO for tuning the hyperparameters in the tailored search space. For simplicity, we define the search space for each cell in the HRT to be the same. For example, if we set the learning rate range between $0.002, 0.8$ with 0.002 increments (400 total learning rates), batch sizes $4, 8, 16$, and local epochs are $5, 10, 15$, the total number of possible combinations of hyperparameters are 3600 per cell.

BO judiciously selects the next points to explore based on the values of the predefined acquisition function obtained from previous exploration steps. We use EI (Expected Improvement) [73] as our acquisition function as it does not require hyperparameter tuning and it is easy for setting intuitive stop conditions. EI aims at maximizing the expected improvement from the new explorations over the current best results and is defined as:

$$EI(Hp) = (y_o - \mu(Hp))\Phi(\gamma(Hp)) + \sigma(Hp)\phi(\gamma(Hp)) \tag{3}$$

where $\mu(\cdot)$ and $\sigma(\cdot)$ are the predictive mean function and predictive standard deviation function, respectively; y_o is the best current value at $argmin_{(Hp)}y(Hp)$; $\gamma(Hp) = \frac{y_o - \mu(Hp)}{Hp}$; $\Phi(\cdot)$ and $\phi(\cdot)$ are predictive cumulative distribution function of standard normal and probability density function of standard normal. *FedCust* also creates a small FL simulator that runs for 20 rounds with 5 clients which we find sufficient. This is done instead of simply evaluating it directly on a single client due our observations of *fairness* as described in Section 3. By tuning on a single client, we risk yielding hyperparameters that overfit on the local dataset, thus reducing generalizability. Instead, by participating in a full FL system, we find that the hyperparameters found are more suited to deriving local models that result in a better overall global model. The dataset heterogeneity properties of these clients are set to those of the cell which the optimizer is running on. The FL simulator returns the final accuracy as the function output, and the BO maximizes this output.

To populate the HRT, *FedCust* uses the BO-based tuner to traverse through each of the possible combinations of quality and dquantity, searching the full hyperparameter space to find the set that gives the highest accuracy. The search for each cell stops when there was no increase in the FL simulator's accuracy in the last n (e.g., 5) rounds. This traversal continues until we find a hyperparameter set for each cell. Therefore, the total search space is number of possible hyperparameters (e.g., 3600) times the number of cells.

We use BO to demonstrate how it can accelerate the construction of the HRT. However, our framework is also compatible with other widely used hyperparameter tuning methods such as Reinforcement Learning and Evolutionary Algorithms. Users are free to select and replace the tuner according to their specific needs.

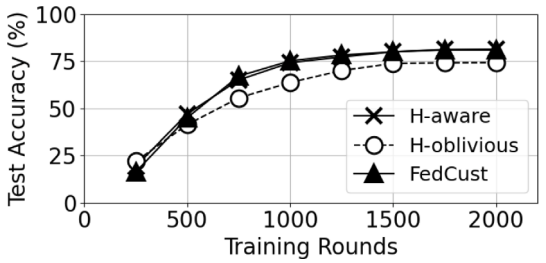


Fig. 6. Accuracy Curve Comparison - The test accuracy comparisons between *FedCust*, Hand-Tuned, and Global Tuning.

Table 3
Accuracy comparison - Accuracy over rounds comparison for different datasets.

Dataset	Global tuning	*FedCust*	Hand-tuning
FEMNIST	74.14%	81.24%	81.64%
Shakespeare	50.99%	54.23%	55.13%
Cifar10	68.13%	72.32/%	72.66%
Cifar100	52.52%	56.21%	56.89%
F-MNIST	73.99%	79.73%	80.03%

6. Evaluation

6.1. Experiment setup

Data Heterogeneity. Due to the lack of production level user datasets, prior literature in FL [11,45,46,54,55] use controlled data distribution heterogeneity. We follow these works for our evaluation as well. The total dataset is split into smaller separate datasets which contains a specific data distribution and quantity heterogeneity (such as *HI* of 0.8 and 800 datapoints) and then assigned to a client (details are provided in Appendix A.4). This is similar to the distribution strategies used in [25,29,54]. Such controlled setups are usually for the purpose of a systematic characterization and clear analysis. It is worth noting that our approach does not assume any specific patterns in data distribution heterogeneity and thus can be applied to any dataset. We also conduct experiments using the Gaussian, Dirichlet and Poisson distributions as used by other papers [45,46], as well as the default distribution used in LEAF [23] to demonstrate that our approach is general and does not depend on specific heterogeneity distribution.

Training and Proxy Dataset Setup. We perform our experiments using the popular image classification datasets FEMNIST [23], Shakespeare [23], Cifar10/Cifar100 [74], and Fashion MNIST [60] (F-MNIST), details in Table A.6. We use the popular FEMNIST dataset for the majority of our demonstrations (in the interest of space) since it was made specifically for benchmarking Federated Learning applications.[2] Since it does not have a separate evaluation dataset, we use the same setup as in [23] and derive a balanced test dataset of size 6200 by randomly sampling 100 datapoints per class from the unused datapoints. Unless otherwise specified, the overall trends are consistent for other datasets too.

The set of proxy datasets are uniformly sampled from their full training datasets. Note that this sampled dataset is removed from the full dataset. Therefore, all proxy datasets have no overlap with either training nor testing datasets. Proxy datasets in FEMNIST, Shakespeare, CIFAR10, CIFAR100, and FashionMNIST (F-MNIST) contain 5000, 15 000, 5000, 5000, and 4000 samples, respectively. The remaining training datasets (after removing the sampled proxy datasets for training) are 44 664, 45 000, 45 000, and 46 000 samples, respectively. We control the different data distributions within each client by splitting them into groups and subgroups. We first create 6 groups of clients by splitting them equally (e.g. in FEMNIST, 192 clients are split into 32 clients per group), and assign each of these devices to get 100/200/400/600/800/1000 data points respectively. We further split these groups into 4 more evenly split subgroups (e.g. in FEMNIST, 32 clients get split into 4 groups of 8). These groups are then assigned varying *HI*s between 0.2 and 1.0.

Hyperparameter Optimization Methods. We compare our BO-based solution to Random Search (method used by *FedEx*) and Grid Search baselines since there are no dedicated hyperparameter tuning frameworks for FL. For Random Search, we perform uniform random sampling from the full search space without replacement and keep the hyperparameter set that gives us the highest accuracy per cell. In Grid Search, we traverse the full space in order and only keep the best hyperparameter set. The total search space is 86,400 possible combinations of hyperparameters, as explained in the previous sections.

Testbed. We build a FL testbed using Tensorflow for the datasets by deploying each client on a cluster with its own exclusive Intel Xeon 2.2 GHz CPU. The server (i.e. aggregator) is deployed on a separate node with 40 CPUs. *FedCust*'s Bayesian Strengthened Tuner, Random Search and Grid Search are performed on the server. The server and the clients communicate their weights via sockets.

[2] https://github.com/TalwalkarLab/leaf/.

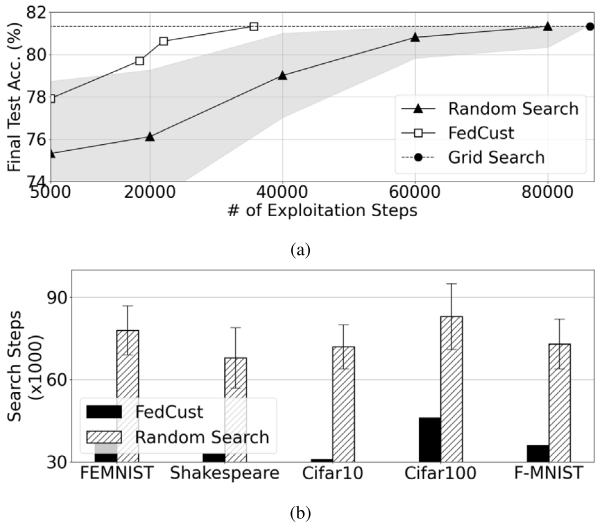


Fig. 7. Cost Comparison - (a) Test accuracy of the global model achieved with hyperparameters derived at different stages of tuning for the FEMNIST dataset. (b) Tuning iterations comparison across different datasets.

Table 4
Accuracy Improvement - Increase in accuracy (%) over Global Tuning for different datasets and distribution metrics.

Dataset	Gaussian ($\beta = 15$)			Poisson ($\lambda = 70$)			Dirichlet ($\alpha = 0.01$)		
	Grid	Random	*FedCust*	Grid	Random	*FedCust*	Grid	Random	*FedCust*
FEMNIST	11.8	8.2	11.1	7.4	4.3	6.7	9.3	7.2	8.9
Cifar10	9.1	4.3	8.8	7.6	2.5	6.8	11.3	8.5	10.3
Shakespeare	3.1	1.2	2.9	1.3	0.4	0.8	2.4	1.7	1.9

6.2. Performance comparison

First, we run different datasets FEMNIST, Shakespeare, Cifar10, Cifar100, and F-MNIST separately with the FL setting. We compare Global Tuning, *FedCust*, and Hand-tuning method, and present the best test accuracy achieved in Table 3. We can find that our proposed method *FedCust* outperforms Global Tuning by a large margin in all models. For example, on FEMNIST dataset, the test accuracy by *FedCust* improves more than 7% than Global Tuning (Fig. 6). On other datasets, *FedCust* still gets better accuracy by at least 3.69% increase. On the other hand, the accuracies achieved with *FedCust* come very close to the Hand-Tuning method with less than 1% margin of error. This demonstrates that our framework can achieve a model performance on par with the best case. Apart from *HI*, state-of-the-art papers also use other methods of quantifying heterogeneity such as *Gaussian*, *Dirichlet* and *Possion* distribution sampling for both quantity and distribution heterogeneity [31,54]. To demonstrate that *FedCust* works with the other distribution metrics, we compare the accuracy increase we get after applying *FedCust* compared to Global Tuning. We do this across the various metrics we discussed in Section 5.2, as shown in Table 4. For this experiment, we generated data quality and quantity for each client with the mean sampling distribution parameters given in the table (set by following literature [62,64,66]). The number of total clients, clients per round and all other setup parameters were kept the same as for the *HI* experiments. The HRT and search space also uses the same cell granularities and search value ranges. Thus, with little changes in the overall BO and HRT configurations, we can achieve reasonably good results. We observe that across all datasets, *FedCust* results in varying degrees of accuracy improvement for all different types of data heterogeneity metrics. *FedCust* incurs significantly less cost for searching than Grid Search yet yields final model improvements within 0.5% using a static HRT granularity and search space. These results demonstrate that *FedCust* is generalizable to heterogeneity metrics or underlying data distributions, and can completely offload hyperparameter tuning to the server. The main idea of simulating data distribution using proxy datasets and tuning on it instead of on local devices is robust to whatever metric is used to measure the data distribution.

6.3. Hyperparameter optimization cost

We next evaluate the efficiency of *FedCust*'s tuner compared to Random Search and Grid Search (Hand-Tuning can be considered as Grid Search). In Fig. 7(a), we show the test accuracies achieved by the FL system when using the hyperparameters found after searching for that particular number of steps for the FEMNIST dataset. Each step represents the number of total hyperparameter combinations searched. We observe that *FedCust*'s Bayesian Strengthened Tuner explores the space efficiently and achieves the same accuracy as the Grid Search method (81.24%) after around 36,000 steps, which is less than half the total search space. For Random

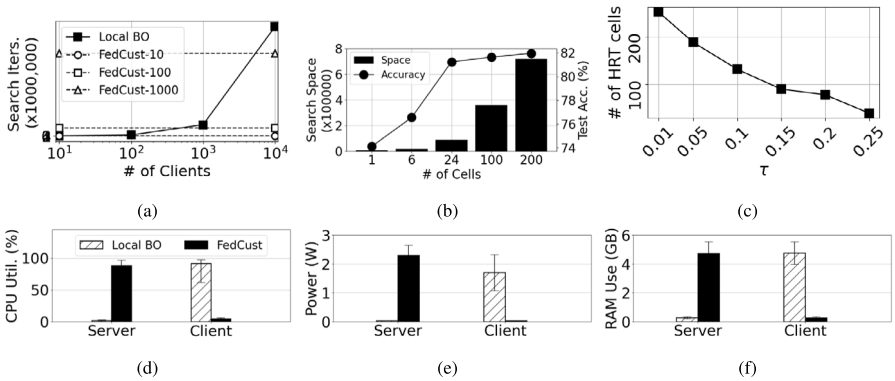


Fig. 8. Scalability of number of clients and HRT and Resource cost - (a) Number of iterations for tuning vs. number of clients in the FL system. This compares the search cost scalability of *FedCust* against using local Bayesian Optimization. (b) The hyperparameter search space as a function of HRT cells against test accuracy and cost using FEMNIST. (c) Analysis on how the HRT granularity varies depending on the τ threshold between the current HRT data distribution combination cells and actual data distribution. (d–f) Resource consumption of the various resources for Local BO and *FedCust* during hyperparameter tuning with the 90th percentile.

Search, we show the mean accuracy vs. steps and their 95% error margin after 10 runs with different seeds. We find that while some runs initially perform better than *FedCust* at around 5000 steps, eventually *FedCust* performs better. The mean number of steps taken by Random Search is around 79,000 steps, which is more than twice as that for *FedCust*. The number of steps taken to achieve terminal accuracy for the other datasets for *FedCust* and Random Search with their error margins are given in Fig. 7(b). We see here that *FedCust* consistently outperforms Random Search across all of them.

6.4. Scalability

Given the large scale of mobile and IoT based Federated Learning [8], an good hyperparameter tuning framework should efficiently scale with the number of participating client devices. To evaluate this, we perform the experiments as shown in Fig. 8(a). Here, we change the number of FL clients present in the system and train them from scratch. We use the LEAF's FEMNIST dataset since it can provide 300,000 clients with their own individual data distributions. Here we compare the number of tuning iterations needed for our *FedCust* against local Bayesian Optimization (Local BO) performed on each individual client. For *FedCust*, we run multiple variations of the search algorithm with different HRT cell sizes labeled as *FedCust*-[$size$]. For Local BO, the black-box function of the Bayesian Optimizer is the local training process and the tunable parameters are the learning rate, batch size and number of local epochs, and we count the number of local optimization steps used until convergence.

As the results show in Fig. 8(a), scaling up the number of clients can significantly increase the number of tuning iterations required to yield a good set of hyperparameters for every device for Local BO. The increase is linear, which is expected due to the number of time the local Bayesian Optimization runs is directly proportional to the number of clients. For *FedCust* search, we observe that the number of tuning iterations remains the same regardless of the number of clients. This is because *FedCust* does not require the knowledge of the local datasets for search and therefore the number of clients is inconsequential to the search process. Instead, we clearly see that the number of cells of the HRT is what impacts the tuning cost, which is because the search algorithm only uses the cells of the HRT for tuning. We observe from these results that our method is scalable with the number of clients and therefore can provide a practical method of hyperparameter tuning for large-scale FL systems.

6.5. HRT size

Fig. 8(b) shows the results of the sensitivity analysis of how the number of cells in the HRT impacts the search space. For example, 1 cell means that only one set of hyperparameters is used to train the full system, i.e., a global tuning set. As we increase the number of cells, there is a drastic increase in the total search space, making it expensive to tune. It also shows how the final test accuracy for FEMNIST changes with varying number of HRT cells for our approach. It is clear that the benefits of increasing the number of cells after 24 diminish greatly while the search space keeps on increasing. Thus, in our experiments, an HRT with 24 cell blocks strikes a good balance between search cost and accuracy. Specifically, we use HIs of $0.2, 0.4, 0.6, 0.8$ and data quantities of $100, 200, 400, 600, 800, 1000$ in our evaluation. We also perform an analysis of how the granularity threshold τ impacts the size of HRT size for the default system described at the start of this section (Fig. 8(c)). Since we are using HI as the data quality metric, we use the Euclidean distance as the distance metric but it functions the same with other data quality metrics and its appropriate distance measures (we remove the results in the interest of space). As we increase the threshold of τ, we observe that the lower the number of HRT cells are generated. This is because with a high threshold, there is larger room for error between the actual client's data distribution and the HRT's data distribution. This results in more coarse-grained tuning since there is a larger difference between the actual fine-tuned hyperparameters and the ones available in the HRT. As we know from Fig. 8(b), with increasing the

Table 5
Final model accuracy - Comparison of *FedCust* against the reported state-of-the-art performance. Missing values are due to them not being reported in the paper.

Dataset	Distr.	Global	*FedCust*	FedEx	FedRL
Cifar10	IID	81.1	83.9	–	**84.3**
	Non-IID	50.4	**53.2**	–	52.9
MNIST	IID	97.8	97.9	–	**98.0**
	Non-IID	94.3	**96.9**	–	95.5
Shakespeare	IID	55.2	**58.1**	57.0	–
	Non-IID	52.2	**55.0**	54.6	–

number of cells results in better fine-tuning and thus better final model performance but at higher search cost. Therefore, the distance threshold τ acts as a trade-off parameter between search cost and final model accuracy, and can be set as per requirement by the users. Additionally, users even can customize the HRT dimensions that they believe are most critical to their hyperparameter tuning process, finer-grained metrics and additional dimensions could potentially contribute to overall performance, however, it is important to carefully consider the trade-off between the size of the HRT and the associated computing and communication overhead.

6.6. Resource cost

We next evaluate the memory, computation and power consumption of *FedCust* compared to the local BO method of tuning. As mentioned above, *FedCust* offloads the complete search process on the server instead of executing them on the clients. Fig. 8 shows the results of memory, computation and power consumption on the mobile devices. For this experiment, we set up our clients on a real Samsung S20 Android device (Qualcomm Snapdragon 855 chipset) and profile the resources using the Android SDK Profiler [49]. Here we clearly see that our *FedCust* has almost 0 resource consumption across all resource types during the hyperparameter tuning phase. The local BO consumes a large amount of resources since it requires the training and evaluation of the deep models for every combination of hyperparameters explored on device, which is significantly more compared to the actual rounds of training conducted when the client is selected. The offloading of the search process on the server by *FedCust* results in more resource consumption on the server side but completely takes off the load from the device-side hardware. This is a extremely beneficial in a mobile or IoT environment where these client devices will already have little available resources [75–77]. Thus, our *FedCust* framework is proficient at conserving resources on the client-side as well as being scalable, privacy-preserving and effective.

6.7. Comparison against state-of-the-art

Few works have tackled the challenge of customized hyperparameter tuning for FL. While some of them tend to violate privacy constraints [37,39], two of the latest works *FedEx* [12,13] (which we name *FedRL* for space) avoid this problem by performing tuning on-device per client. Apart from the resource cost, they also have other drawbacks as pointed out in Sections Section 2. We compare the results of FL training using our hyperparameter tuning against theirs, and the results are shown in Table 5.

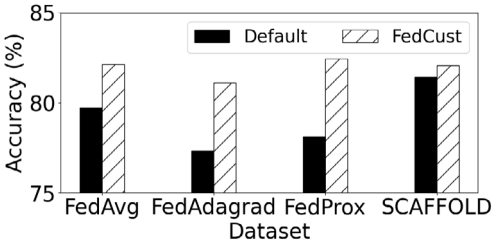
Neither of them have their code open-sourced, so we report their numbers from the paper. We set up our FL system as closely to their systems (i.e. the number of clients, clients per round and per-client data distribution) based on their descriptions. We then use *FedCust* and determine the HRT size using the mechanism defined in Section 5.3. For Cifar10 and MNIST, we observe that our method outperforms *FedRL* for the non-IID case. For the IID case, we are on par. This is due to data heterogeneity being a central theme in our solution and IID data will not effect our tuning customization much. For Shakespeare, we use outperform in both situations since *FedEx*'s focus is more on finding good models and ignore data heterogeneity and hyperparameter tuning mostly.

6.8. Compatibility with other heterogeneity-aware FL optimization

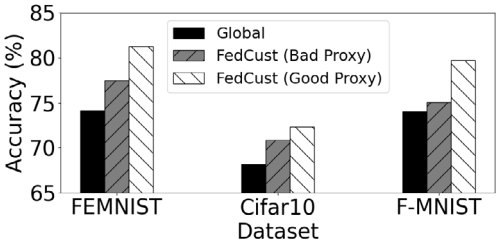
We perform additional experiments to demonstrate *FedCust* is compatible with other state-of-the-art heterogeneity-aware optimizations in FL. In Fig. A.12, the first set of bars shows the comparison of test accuracy at convergence between global tuning (*Default*) and *FedCust* when using LEAF's default distribution. We observe that using our customized hyperparameter tuning can achieve an accuracy improvement of around 2.3%. The second set of bars show the change in accuracy when using FedAdagrad [31] by itself (*Default*) versus adding *FedCust* on top of it (*FedCust*). We observe that with the help of *FedCust*, the final accuracy is improved around 4%, confirming that *FedCust* and FedAdagrad are complementary to each other and can be combined to achieve an even better performance.

6.9. Impact of proxy dataset quality

For the proxy dataset, we assume it contains no knowledge about the training data. In this section, we evaluate the extreme case where the proxy dataset is completely different from the training data. We emulate this situation by using a completely different dataset that has the same number of classes as the proxy dataset to generate the HRT and the accuracy results are present in Fig. 9(b). We observe that in such an extreme scenario, the accuracy indeed drops compared to using a better proxy dataset. However, our approach still outperforms the one-size-fits-all baseline. This verifies that a better quality proxy dataset would indeed improve the model performance, but our method is robust even with a very poor quality proxy dataset.



(a)



(b)

Fig. 9. Compatibility with other frameworks, robustness of proxy datasets. (a) Final test accuracy of state-of-the-art FL frameworks when used with and without *FedCust*. (b) Test accuracy comparison of one-size-fits-all approach (labeled as *Global*), *FedCust* with extremely poor quality of proxy dataset (i.e., use Double MNIST, Cifar100 and MNIST as proxy dataset for FEMNIST, Cifar10 and F-MNIST as training dataset labeled as *FedCust (Bad Proxy)*), and *FedCust* using the same datasets for proxy and training, though no overlapping between proxy and training labeled as *FedCust (Good Proxy)*.

7. Conclusion

In this paper, we provide new insights for efficient hyperparameter customization in FL by identifying the opportunities and challenges via empirical experiments. We observe that the hyperparameter choices vary depending on data heterogeneity and we can group clients based on heterogeneity to share hyperparameters. Inspired by our study, we propose *FedCust*, a privacy preserving and data heterogeneity-aware hyperparameter customization framework for FL which customizes hyperparameters for clients on the server side to avoid imposing overheads on client devices. The core of *FedCust* is the use of heterogeneity measurement metrics for clustering clients into heterogeneity groups and a sever-side proxy dataset based hyperparameter customization approach for addressing the privacy and tuning cost challenges. We evaluate *FedCust* in a real testbed and show that it outperforms baselines and state-of-the-art methods. Therefore, *FedCust* is an effective, privacy-preserving, scalable, and robust hyperparameter customization framework for FL that incurs no additional computational cost on client devices.

CRediT authorship contribution statement

Syed Zawad: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Xiaolong Ma:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Jun Yi:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Cheng Li:** Writing – review & editing, Writing – original draft, Resources, Methodology, Formal analysis, Conceptualization. **Minjia Zhang:** Writing – review & editing, Writing – original draft, Software, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Lei Yang:** Conceptualization, Supervision. **Feng Yan:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Yuxiong He:** Supervision, Software, Resources, Project administration, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by NSF CAREER-2305491.

Table A.6
Training setup.

Dataset	Model	Train/Test split	Clients total/Per round	Global LR/Batch size	Training rounds
FEMNIST	2 conv 2 dense	49,644/6,200	192/10	0.004/8	2000
Cifar100	Resnet18	50,000/10,000	50/5	0.045/16	1000
Cifar10	4 conv 2 dense	50,000/10,000	50/5	0.05/16	500
F-MNIST	2 conv 2 dense	50,000/10,000	50/5	0.002/8	500

Table A.7
HRT - Sample HRT for HI data quality metric. Each cell contains the tuned learning rate, batch size, local epochs for that distribution combination. The *HI* can be substituted with any other heterogeneity metric, and is shown as an example only.

		# of data points			
		100	200	.....	1000
HI	0.2	2e−4/8/4	5e−4/8/6	–	5e−4/8/20
	0.4	4e−4/8/15	2e−3/6/20	–	2e−3/24/20
	....	–	–	–	–
	1.0	4e−3/8/8	4e−3/16/20	–	4e−3/16/30

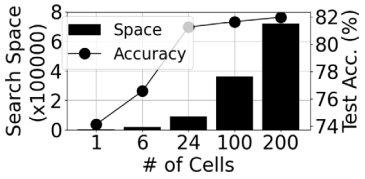


Fig. A.10. Sensitivity analysis - The hyperparameter search space as a function of HRT cells against test accuracy and cost using FEMNIST.

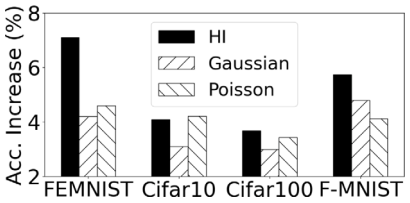


Fig. A.11. Final test accuracy - Comparison between global, *FedCust* with transferred dataset and *FedCust* on original dataset.

Appendix. Supplementary materials

A.1. Sensitivity analysis of HRT size against accuracy

Fig. A.10 shows the results of the sensitivity analysis of how the number of cells in the HRT impacts the search space. For example, 1 cell means that only one set of hyperparameters is used to train the full system, i.e., a global tuning set. As we increase the number of cells, there is a drastic increase in the total search space, making it expensive to tune. Fig. A.10 shows how the final test accuracy for FEMNIST changes with varying number of HRT cells for our approach. It is clear that the benefits of increasing the number of cells after 24 diminish greatly while the search space keeps on increasing. Thus, in our experiments, an HRT with 24 cell blocks strikes a good balance between search cost and accuracy. Specifically, we use HIs of $0.2, 0.4, 0.6, 0.8$ and data quantities of $100, 200, 400, 600, 800, 1000$ in our evaluation.

A.2. Robustness to data heterogeneity metrics

Apart from *HI*, state-of-the-art papers also use other methods of quantifying heterogeneity such as *Gaussian* and *Possion* distribution sampling (for both quantity and distribution heterogeneity) [31,54]. To demonstrate that *FedCust* works with other distribution metrics, we compare the accuracy increase we get after applying *FedCust* compared to Global Tuning. We do this across the heterogeneity types *HI*, *Gaussian*, and *Poisson*, and the results are presented in Fig. A.11. We observe that across all datasets, *FedCust* results in varying degrees of accuracy improvement for all different types of data heterogeneity metrics. This demonstrates that our framework is robust to different types of data distribution metrics.

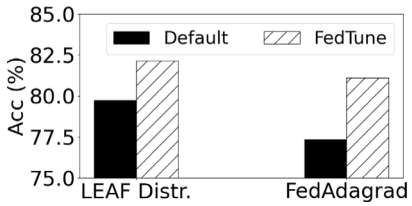


Fig. A.12. Final test accuracy comparison - Comparison between global and *FedCust* when using LEAF's [23] default distribution (*LEAF Distr.*) and when used with and without *FedCust*.

A.3. Compatibility with other heterogeneity-aware FL optimization

We perform additional experiments to demonstrate *FedCust* is compatible with other state-of-the-art heterogeneity-aware optimizations in FL. In Fig. A.12, the first set of bars show the comparison of test accuracy at convergence between global tuning (*Default*) and *FedCust* when using LEAF's default distribution. We observe that using our customized hyperparameter tuning can achieve an accuracy improvement of around 2.3%. The second set of bars show the change in accuracy when using FedAdagrad [31] by itself (*Default*) versus adding *FedCust* on top of it (*FedCust*). We observe that with the help of *FedCust*, the final accuracy is improved around 4%, confirming that *FedCust* and FedAdagrad are complementary to each other and can be combined to achieve an even better performance.

A.4. Training and proxy dataset setup

We perform our experiments using the popular image classification datasets Cifar100, Cifar10 and Fashion-MNIST. We also use the widely used FEMNIST dataset, which is a handwritten digit and character image classification dataset made specifically for benchmarking Federated Learning applications. It contains 62 classes and around 800,000 images split into 3550 clients. We sample from it using the seed and sample found in their official repository in github[3] and our code.[4] Since it does not have a separate evaluation dataset, we use the same setup in [23] and derive a balanced test dataset of size 6200 by randomly sampling 100 datapoints per class from the unused datapoints.

The set of proxy datasets are uniformly sampled from their full training datasets. Note that this sampled dataset is removed from the full dataset. Therefore, all proxy datasets have no overlap with either training nor testing datasets. Proxy datasets in FEMNIST, CIFAR10, CIFAR100, and FashionMNIST contain 5000, 5000, 5000, and 4000 samples, respectively. The remaining training datasets (after removing the sampled proxy datasets for training) are 44 664, 45 000, 45 000, and 46 000 samples, respectively.

We control the different data distributions within each client by splitting them into groups and subgroups. We first create 6 groups of clients by splitting them equally (e.g. in FEMNIST, 192 clients are split into 32 clients per group), and assign each of these devices to get 100/200/400/600/800/1000 data points respectively. We further split these groups into 4 more evenly split subgroups (e.g. in FEMNIST, 32 clients get split into 4 groups of 8). These groups are then assigned *HI*s of 0.2, 0.4, 0.6 and 0.8.

References

[1] J. Konečný, H.B. McMahan, F.X. Yu, P. Richtárik, A.T. Suresh, D. Bacon, Federated learning: Strategies for improving communication efficiency, 2016, arXiv preprint arXiv:1610.05492.
[2] V. Smith, C.-K. Chiang, M. Sanjabi, A. Talwalkar, Federated multi-task learning, 2017, arXiv preprint arXiv:1705.10467.
[3] J. Liang, S. Li, B. Cao, W. Jiang, C. He, Omnilytics: A blockchain-based secure data market for decentralized machine learning, 2021, arXiv preprint arXiv:2107.05252.
[4] Z. Zhang, R. Hu, Byzantine-robust federated learning with variance reduction and differential privacy, in: 2023 IEEE Conference on Communications and Network Security, CNS, IEEE, 2023, pp. 1–9.
[5] T. Li, A.K. Sahu, A. Talwalkar, V. Smith, Federated learning: Challenges, methods, and future directions, IEEE Signal Process. Mag. 37 (3) (2020) 50–60.
[6] Y. Wang, D. Kumar, A. Chandra, Poster: Exploiting data heterogeneity for performance and reliability in federated learning, in: 2020 IEEE/ACM Symposium on Edge Computing, SEC, IEEE, 2020, pp. 164–166.
[7] T. Li, A.K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, V. Smith, Federated optimization in heterogeneous networks, 2018, arXiv preprint arXiv:1812.06127.
[8] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, et al., Towards federated learning at scale: System design, Proc. Mach. Learn. Syst. 1 (2019) 374–388.
[9] H. Wang, Z. Kaplan, D. Niu, B. Li, Optimizing federated learning on non-iid data with reinforcement learning, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, 2020, pp. 1698–1707.
[10] C. Yang, Q. Wang, M. Xu, Z. Chen, K. Bian, Y. Liu, X. Liu, Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data, 2021, arXiv:2006.06983.
[11] H.B. McMahan, et al., Advances and open problems in federated learning, Found. Trends® Mach. Learn. 14 (1) (2021).
[12] M. Khodak, T. Li, L. Li, M. Balcan, V. Smith, A. Talwalkar, Weight sharing for hyperparameter optimization in federated learning, in: Int. Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with ICML 2020, 2020.

[3] https://github.com/TalwalkarLab/leaf/.
[4] https://anonymous.4open.science/r/FLTUNE-EAD4/.

[13] H. Mostafa, Robust federated learning through representation matching and adaptive hyper-parameters, 2019, arXiv e-prints, arXiv–1912.
[14] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (2) (2012).
[15] M.A. Wiering, M. Van Otterlo, Reinforcement learning, Adapt. Learn. Optim. 12 (3) (2012) 729.
[16] S. Savazzi, M. Nicoli, V. Rampa, Federated learning with cooperating devices: A consensus approach for massive IoT networks, IEEE Internet Things J. 7 (5) (2020) 4641–4654.
[17] D.C. Nguyen, M. Ding, P.N. Pathirana, A. Seneviratne, J. Li, H.V. Poor, Federated learning for internet of things: A comprehensive survey, IEEE Commun. Surv. Tutor. 23 (3) (2021) 1622–1658.
[18] G. Lan, X.-Y. Liu, Y. Zhang, X. Wang, Communication-efficient federated learning for resource-constrained edge devices, IEEE Trans. Mach. Learn. Commun. Netw. (2023).
[19] A. Fallah, A. Mokhtari, A. Ozdaglar, Personalized federated learning: A meta-learning approach, 2020, arXiv preprint arXiv:2002.07948.
[20] A.Z. Tan, H. Yu, L. Cui, Q. Yang, Towards personalized federated learning, IEEE Trans. Neural Netw. Learn. Syst. (2022).
[21] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H.B. McMahan, S. Patel, D. Ramage, A. Segal, K. Seth, Practical secure aggregation for privacy-preserving machine learning, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1175–1191.
[22] S. Wang, T. Tuor, T. Salonidis, K.K. Leung, C. Makaya, T. He, K. Chan, Adaptive federated learning in resource constrained edge computing systems, IEEE J. Sel. Areas Commun. 37 (6) (2019) 1205–1221.
[23] S. Caldas, S.M.K. Duddu, P. Wu, T. Li, J. Konečnỳ, H.B. McMahan, V. Smith, A. Talwalkar, Leaf: A benchmark for federated settings, 2018, arXiv preprint arXiv:1812.01097.
[24] A. Ghosh, J. Hong, D. Yin, K. Ramchandran, Robust federated learning in a heterogeneous environment, 2019, arXiv preprint arXiv:1906.06629.
[25] X. Li, K. Huang, W. Yang, S. Wang, Z. Zhang, On the convergence of FedAvg on non-IID data, in: International Conference on Learning Representations, 2019.
[26] G. Ding, Z. Li, Y. Wu, X. Yang, M. Aliasgari, H. Xu, Towards an efficient client selection system for federated learning, in: International Conference on Cloud Computing, Springer, 2022, pp. 13–21.
[27] J. Qian, X. Fafoutis, L.K. Hansen, Towards federated learning: Robustness analytics to data heterogeneity, 2020, arXiv preprint arXiv:2002.05038.
[28] J. Wolfrath, N. Sreekumar, D. Kumar, Y. Wang, A. Chandra, Haccs: Heterogeneity-aware clustered client selection for accelerated federated learning, in: 2022 IEEE International Parallel and Distributed Processing Symposium, IPDPS, IEEE, 2022, pp. 985–995.
[29] Z. Chai, A. Ali, S. Zawad, S. Truex, A. Anwar, N. Baracaldo, Y. Zhou, H. Ludwig, F. Yan, Y. Cheng, Tifl: A tier-based federated learning system, in: Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing, 2020, pp. 125–136.
[30] S.P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, A.T. Suresh, SCAFFOLD: Stochastic controlled averaging for federated learning, in: International Conference on Machine Learning, PMLR, 2020, pp. 5132–5143.
[31] S.J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, H.B. McMahan, Adaptive federated optimization, in: International Conference on Learning Representations, 2020.
[32] T. Lin, L. Kong, S.U. Stich, M. Jaggi, Ensemble distillation for robust model fusion in federated learning, 2020, arXiv preprint arXiv:2006.07242.
[33] K. Pillutla, S.M. Kakade, Z. Harchaoui, Robust aggregation for federated learning, 2019, arXiv preprint arXiv:1912.13445.
[34] X. Yao, T. Huang, R.-X. Zhang, R. Li, L. Sun, Federated learning with unbiased gradient aggregation and controllable meta updating, 2019, arXiv preprint arXiv:1910.08234.
[35] H. Zheng, M. Gao, Z. Chen, X. Feng, A distributed hierarchical deep computation model for federated learning in edge computing, IEEE Trans. Ind. Inform. (2021).
[36] W. Wu, L. He, W. Lin, R. Mao, Accelerating federated learning over reliability-agnostic clients in mobile edge computing systems, IEEE Trans. Parallel Distrib. Syst. (2020).
[37] Z. Dai, B.K.H. Low, P. Jaillet, Federated Bayesian optimization via thompson sampling, Adv. Neural Inf. Process. Syst. 33 (2020).
[38] Z. Dai, B.K.H. Low, P. Jaillet, Differentially private federated Bayesian optimization with distributed exploration, Adv. Neural Inf. Process. Syst. (2021).
[39] S. Holly, T. Hiessl, S.R. Lakani, D. Schall, C. Heitzinger, J. Kemnitz, Evaluation of hyperparameter-optimization approaches in an industrial federated learning system, 2021, arXiv preprint arXiv:2110.08202.
[40] Y. Zhou, P. Ram, T. Salonidis, N. Baracaldo, H. Samulowitz, H. Ludwig, FLoRA: Single-shot hyper-parameter optimization for federated learning, 2021, arXiv preprint arXiv:2112.08524.
[41] J. Wang, Q. Liu, H. Liang, G. Joshi, H.V. Poor, Tackling the objective inconsistency problem in heterogeneous federated optimization, Adv. Neural Inf. Process. Syst. 33 (2020).
[42] H. Zhang, M. Zhang, X. Liu, P. Mohapatra, M. DeLucia, Automatic tuning of federated learning hyper-parameters from system perspective, 2021, arXiv preprint arXiv:2110.03061.
[43] S. Agrawal, S. Sarkar, M. Alazab, P.K.R. Maddikunta, T.R. Gadekallu, Q.-V. Pham, Genetic CFL: hyperparameter optimization in clustered federated learning, Comput. Intell. Neurosci. 2021 (2021).
[44] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1273–1282.
[45] C. Briggs, Z. Fan, P. Andras, Federated learning with hierarchical clustering of local updates to improve training on non-IID data, in: 2020 International Joint Conference on Neural Networks, IJCNN, IEEE, 2020, pp. 1–9.
[46] F. Sattler, S. Wiedemann, K.-R. Müller, W. Samek, Robust and communication-efficient federated learning from non-iid data, IEEE Trans. Neural Netw. Learn. Syst. 31 (9) (2019) 3400–3413.
[47] T. Nishio, R. Yonetani, Client selection for federated learning with heterogeneous resources in mobile edge, in: ICC 2019-2019 IEEE International Conference on Communications, ICC, IEEE, 2019, pp. 1–7.
[48] S.L. Smith, P.-J. Kindermans, C. Ying, Q.V. Le, Don't decay the learning rate, increase the batch size, 2017, arXiv preprint arXiv:1711.00489.
[49] G. Inc, The android profiler, 2022, URL https://developer.android.com/studio/profile/android-profiler.
[50] G. Callebaut, G. Leenders, J. Van Mulders, G. Ottoy, L. De Strycker, L. Van der Perre, The art of designing remote IoT devices—Technologies and strategies for a long battery life, Sensors 21 (3) (2021) 913.
[51] A. Zouinkhi, A. Flah, L. Mihet-Popa, A novel energy-safe algorithm for enhancing the battery life for IoT sensors' applications, Energies 14 (20) (2021) 6613.
[52] Y. Guo, Z. Zhao, K. He, S. Lai, J. Xia, L. Fan, Efficient and flexible management for industrial internet of things: A federated learning approach, Comput. Netw. 192 (2021) 108122.
[53] A. Imteaj, U. Thakker, S. Wang, J. Li, M.H. Amini, A survey on federated learning for resource-constrained IoT devices, IEEE Internet Things J. (2021).
[54] S. Zawad, A. Ali, P.-Y. Chen, A. Anwar, Y. Zhou, N. Baracaldo, Y. Tian, F. Yan, Curse or redemption? How data heterogeneity affects the robustness of federated learning, 2021, arXiv preprint arXiv:2102.00655.
[55] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, V. Chandra, Federated learning with non-iid data, 2018, arXiv preprint arXiv:1806.00582.
[56] N. Guha, A. Talwalkar, V. Smith, One-shot federated learning, 2019, arXiv preprint arXiv:1902.11175.
[57] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, F. Beaufays, Applied federated learning: Improving google keyboard query suggestions, 2018, arXiv preprint arXiv:1812.02903.

[58] H. Zhang, J. Bosch, H.H. Olsson, Engineering federated learning systems: A literature review, in: International Conference on Software Business, Springer, 2020, pp. 210–218.
[59] Y. LeCun, The MNIST database of handwritten digits, 1998, http://yann.lecun.com/exdb/mnist/.
[60] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017, arXiv preprint arXiv:1708.07747.
[61] A. Reisizadeh, F. Farnia, R. Pedarsani, A. Jadbabaie, Robust federated learning: The case of affine distribution shifts, Adv. Neural Inf. Process. Syst. 33 (2020) 21554–21565.
[62] Z. Sun, P. Kairouz, A.T. Suresh, H.B. McMahan, Can you really backdoor federated learning? 2019, arXiv preprint arXiv:1911.07963.
[63] N. Agarwal, P. Kairouz, Z. Liu, The skellam mechanism for differentially private federated learning, Adv. Neural Inf. Process. Syst. 34 (2021) 5052–5064.
[64] T. Chen, X. Jin, Y. Sun, W. Yin, Vafl: a method of vertical asynchronous federated learning, 2020, arXiv preprint arXiv:2007.06081.
[65] Q. Li, B. He, D. Song, Model-contrastive federated learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 10713–10722.
[66] T. Li, S. Hu, A. Beirami, V. Smith, Ditto: Fair and robust federated learning through personalization, in: International Conference on Machine Learning, PMLR, 2021, pp. 6357–6368.
[67] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, D. Ramage, Federated learning for mobile keyboard prediction, 2018, arXiv preprint arXiv:1811.03604.
[68] M.T. Pilehvar, J. Camacho-Collados, Embeddings in natural language processing: Theory and advances in vector representations of meaning, Synth. Lect. Hum. Lang. Technol. 13 (4) (2020) 1–175.
[69] R.L. Plackett, Karl Pearson and the chi-squared test, Int. Statist. Rev./Rev. Int. Statist. (1983) 59–72.
[70] T. Kailath, The divergence and bhattacharyya distance measures in signal selection, IEEE Trans. Commun. Technol. 15 (1) (1967) 52–60.
[71] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, R. Adams, Scalable Bayesian optimization using deep neural networks, in: International Conference on Machine Learning, PMLR, 2015, pp. 2171–2180.
[72] J. Wu, S. Toscano-Palmerin, P.I. Frazier, A.G. Wilson, Practical multi-fidelity Bayesian optimization for hyperparameter tuning, in: Uncertainty in Artificial Intelligence, PMLR, 2020, pp. 788–798.
[73] E. Vazquez, J. Bect, Convergence properties of the expected improvement algorithm with fixed mean and covariance functions, J. Statist. Plann. Inference 140 (11) (2010) 3088–3095.
[74] A. Krizhevsky, et al., Learning Multiple Layers of Features from Tiny Images, Technical Report, University of Toronto, 2009.
[75] W.Y.B. Lim, N.C. Luong, D.T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, C. Miao, Federated learning in mobile edge networks: A comprehensive survey, IEEE Commun. Surv. Tutor. 22 (3) (2020) 2031–2063.
[76] C. Zhan, M. Ghaderibaneh, P. Sahu, H. Gupta, Deepmtl: Deep learning based multiple transmitter localization, in: 2021 IEEE 22nd International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM, IEEE, 2021, pp. 41–50.
[77] M. Aledhari, R. Razzak, R.M. Parizi, F. Saeed, Federated learning: A survey on enabling technologies, protocols, and applications, IEEE Access 8 (2020) 140699–140725.