FedPerf: A Practitioners' Guide to Performance of Federated Learning Algorithms

Ajinkya Mulay[*] MULAY@PURDUE.EDU
Purdue University

Baye Gaspard[*] B.GASPARD@INNOPOLIS.RU
Innopolis University

Rakshit Naidu[*] RNEMAKAL@ANDREW.CMU.EDU
Manipal Institute of Technology, Carnegie Mellon University

Santiago Gonzalez-Toral[*] HERNAN.GONZALEZT@UCUENCA.EDU.EC
University of Cuenca

Vineeth S[*] VINEETHS@IISC.AC.IN
Indian Institute of Science

Tushar Semwal[*] TUSHARSEMWAL@OUTLOOK.COM
The University of Edinburgh, Mobius Labs GmbH

Ayush Manish Agrawal[†] AAGRAWAL@NEBRASKA.EDU
University of Nebraska-Lincoln

Abstract

Federated Learning (FL) enables edge devices to collaboratively train a global model without sharing their local data. This decentralized and distributed approach improves user privacy, security, and trust. Different variants of FL algorithms have presented promising results on both IID and skewed Non-IID data. However, the performance of FL algorithms is found to be sensitive to the FL system parameters and hyperparameters of the used model. In practice, tuning the right set of parameter settings for an FL algorithm is an expensive task. In this preregister paper, we propose an empirical investigation on four prominent FL algorithms to discover the relation between the FL System Parameters (FLSPs) and their performances. The FLSPs add extra complexity to FL algorithms over a traditional ML system. We hypothesize that choosing the best FL algorithm for the given FLSP is not a trivial problem. Further, we endeavor to formulate a systematic method that could aid the practitioners in selecting a suitable algorithm given the FLSPs. The code for all the experiments is available here: https://github.com/tushar-semwal/fedperf.

Keywords: Federated Learning, Privacy-AI, Distributed Machine Learning, Sensitivity Analyses

[*] All co-authors except the last (Ayush Manish Agrawal) equally contributed. All authors are also associated with OpenMined.

[†] Least contribution. The co-author did not contribute to the results section in this paper.

© 2021 A. Mulay, B. Gaspard*, R. Naidu*, S. Gonzalez-Toral*, V. S*, T. Semwal* & A.M. Agrawal.

1. Introduction

Data is naturally found to be decentralized and distributed across edge devices. Conventional Machine Learning (ML) approaches involve first collecting this data into a central server and then training a global model using the aggregated dataset. Though this centralized form of training provides better control, however, it suffers from two paramount issues. The first is the privacy of the data owners as governed by the General Data Protection Regulation (Voigt and Von dem Bussche, 2017) and Health Insurance Portability and Accountability Act (Annas, 2003). The other major concern with traditional ML approaches is the communication overhead. For instance, uploading the data from a resource-limited end-device to a central server depletes the battery.

Federated Learning (FL) (McMahan et al., 2017) is a new paradigm that allows the edge devices called *clients* to train a global model collaboratively. FL involves multiple communication cycles. In each cycle, a set of clients train an ML model on their local data and share only the model updates (gradients) with the central server. The central server then aggregates these updates from a pool of selected clients and does a single update to the global model. Finally, the server shares the updated global model back to the clients, thereby completing one cycle. Thus, instead of sharing the data, the clients only share the marginally smaller-sized model updates, hence reducing communication overheads and avoiding a potential privacy leak.

Since its inception, FL has shown promising results on training decentralized and Non-Independent and Identically Distributed (Non-IID) datasets. Hard et al. (2018) introduced a recurrent neural network model for the next-word prediction task in Google virtual keyboard (GBoard) on millions of mobile devices. A similar large-scale system design of FL is described in (Bonawitz et al., 2019). Nevertheless, FL has its own set of critical challenges and downsides, especially in scenarios where deep learning models are used. The performance of an FL algorithm is found to be highly sensitive to both the system- and hyper-parameters of the model (for instance, a deep neural network) (McMahan et al., 2017). In practice, exploring the right set of configuration settings for an FL algorithm is a costly and arduous task. The primary reason being that training FL algorithms in a simulation setting is much slower than the conventional DL approaches since it involves training multiple models sequentially.

In this paper, we present `FedPerf`, an empirical study on four prominent FL algorithms to capture the relation between the parameters and performance. To provide for the unique scenario created by FL, we define Federated Learning System Parameters (FLSPs). These FLSPs are an added complexity in FL over a traditional ML system. For example, one significant FLSP is the skewness in the Non-IID data distributed among the client devices. A highly skewed data could comprise a scenario with each client containing only a single label data in a multi-class classification problem. Other common FLSPs include the number of participating clients, the communication and processing budget of individual clients, and the fraction of stragglers. We are inspired by the previous work from Zhang and Wallace (2017), and Semwal et al. (2018), which presents a similar analysis on convolutional neural networks and Transfer Learning (Pan and Yang, 2009), respectively. It is envisaged that this work will significantly reduce the efforts of practitioners and researchers expend in finding the proper settings. Here, we will explore and report the results of a large set of experiments on four different state-of-the-art FL algorithms. Many of the recent work in FL either present a conceptual survey or only discuss the mean accuracies for their own set of selected hyperparameters values. However, we found that the performance of FL algorithms is highly susceptible to the choice of constant parameters. We hypothesize that choosing the suitable FL algorithm and hyperparameters for the given FLSPs while tuning the FLSPs is not a trivial task. Furthermore, we are interested in identifying the inherent limitations of the FL system. Developing a thorough understanding of the FL system will help us lay the groundwork for expanding FL with Secure Multi-Party Computation (SMPC), Homomorphic Encryption (HE), and Differential Privacy (DP).

We explore the following aspects of FL:

1. How do we characterize the effect of FLSPs on the performance of an FL system?
2. Given FLSPs for a system, can we identify the best performing FL algorithm?
3. How can we use a structured approach to allow practitioners to systematically choose a suitable FL algorithm given the FLSPs?

2. Related Work

Lim et al. (2020) discuss challenges in FL that occur in highly heterogeneous systems. These include communication costs, resource allocation, privacy, and security in the implementation of FL at scale. Similarly, Aledhari et al. (2020) provide industry-specific obstacles in FL, along with detailed service use-cases. However, both of these manuscripts do not address the complexity involved in implementing and comparing different FL algorithms. Keeping these papers in mind, we further investigate the motivation behind FLSPs and Federated Learning Metric (FLM).

FLSPs: Understanding the ever-increasing list of system parameters, which affect FL, is hard to keep track of. Further, investigating every single FL system configuration is virtually and computationally infeasible. We do see an attempt in Hu et al. (2020) to determine the important parameters in FL systems. They investigate unique parameters like dataset partitioning styles and the diversity of datasets. Furthermore, the authors have made attempts to present metrics that could track these parameters separately. Similarly, in Li et al. (2019), we see a new parameter called *client fairness* being discussed, thus prompting the idea that there could be more undiscovered parameters that might be important and yet not brought into the discussions. In, (Nilsson et al., 2018), the authors compare the performance of multiple FL algorithms for one FLSP – data distribution heterogeneity (either IID or non-IID). Based on the performance analysis reported in (Nilsson et al., 2018), the number of simulations for m algorithms, and a single FLSP, would be in the order of $O(mn)$ where m is the number of FL algorithms and n is the different variations of each FLSP. Thus, exhibiting the complexity of choosing FLSPs. Fig. 1 further showcases the complexity in tuning multiple FLSPs along with hyperparameters for an FL system. As can be seen from the figure, the number of experiments increases polynomial with the increase in the FLSPs. We believe understanding the relationship between FLSPs and the performance of FL Systems could help reduce the time required to simulate FL algorithms.

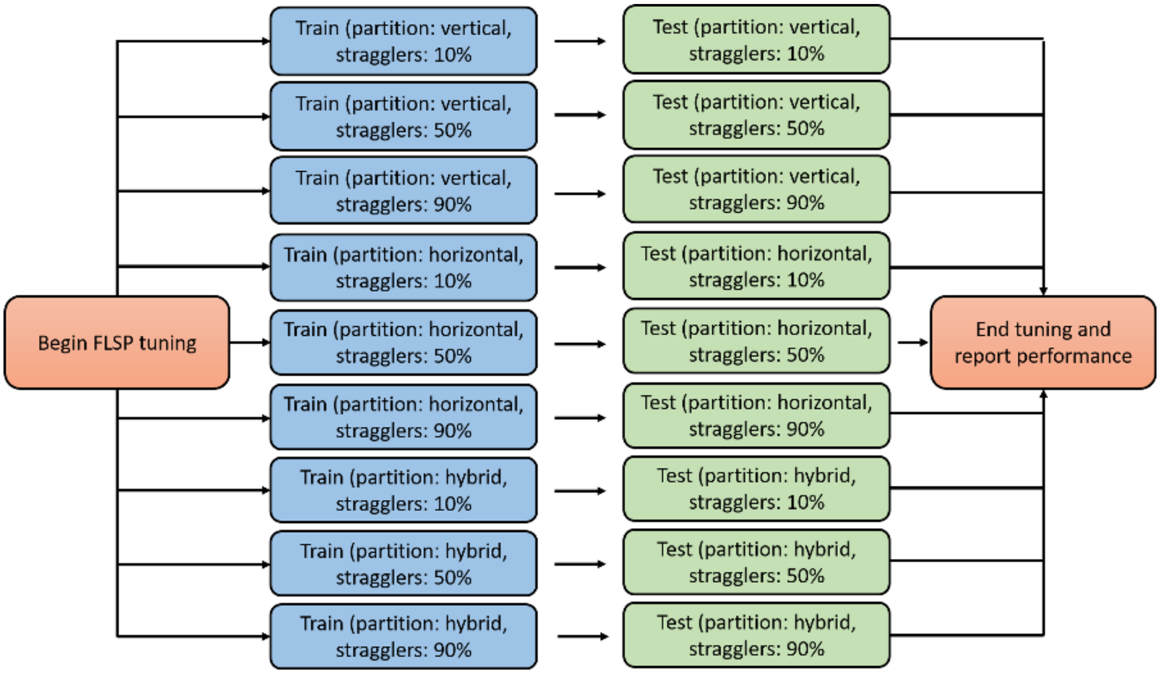

Begin FLSP tuning
Train (partition: vertical, stragglers: 10%
Train (partition: vertical, stragglers: 50%
Train (partition: vertical, stragglers: 90%
Train (partition: horizontal, stragglers: 10%
Train (partition: horizontal, stragglers: 50%
Train (partition: horizontal, stragglers: 90%
Train (partition: hybrid, stragglers: 10%
Train (partition: hybrid, stragglers: 50%
Train (partition: hybrid, stragglers: 90%
Test (partition: vertical, stragglers: 10%
Test (partition: vertical, stragglers: 50%
Test (partition: vertical, stragglers: 90%
Test (partition: horizontal, stragglers: 10%
Test (partition: horizontal, stragglers: 50%
Test (partition: horizontal, stragglers: 90%
Test (partition: hybrid, stragglers: 10%
Test (partition: hybrid, stragglers: 50%
Test (partition: hybrid, stragglers: 90%
End tuning and report performance

Figure 1: **FLSP Tuning**: A graphical demonstration of the complexity of FLSP tuning to identify the performance of FL algorithms. Here, we have to discretise continuous FLSP spaces necessarily, (the percentage of stragglers in the system) to reduce the parameter search complexity. Even then, we have nine configurations. Furthermore, adding in the hyperparameter search leads to a doubling of the parameter tuning complexity.

In this paper, we therefore, attempt to provide a broad definition inclusive of all current and potential future FLSPs. Furthermore, we provide comprehensive coverage of FLSPs known through the literature and our findings. Finally, we provide an empirical correlation between FLPSs and FL performance.

FLM: As reported in Hu et al. (2020), tracking performance is increasingly complex since there could be multiple metrics for the *same* FL system. Liang et al. (2020) provide an industry-specific empirical coverage of the parameters that are valuable in real datasets, such as algorithm robustness and fairness between corporations. As evidenced above, the literature is not yet complete enough to provide simple easy-to-track metrics in FL. This paper attempts to provide clarity on these metrics. We investigate how we can reduce the number of metrics to be tracked and develop a simple easy-to-track metric (denoted by FLM). In this paper, we experiment on five standard Federated Learning protocols with various FLSPs.

Table 1: FLSP: List of system parameters which affect the performance of FL algorithms

FLSP	Types
Datasets	Standard benchmark datasets
Data Variety	Images and text
Data Skewness	Difference in data size and number of labels across clients
Data distribution heterogeneity	IID and Non-IID
Communication	Number of global rounds
Number of clients	Variations over possible number of clients
Stragglers	Percentage varying from 0%-95%
ML Models	Choosing multiple ML models appropriate for the task
Synchronicity	Synchronous and asynchronous
Client Fairness	Maximum performance difference (in %) the over clients
Computational power	Number of local rounds

3. Methodology and experimental protocol

Since we wish to understand the impact of FLSPs on the system performance, we will begin with a simple experiment over baseline federated datasets used in (McMahan et al., 2017). Table 1 presents a list of currently identified FLSPs. We tune the four algorithms with respect to the FLSPs, by tuning one FLSP at a time while keeping the rest constant. The algorithm performance is measured over the metrics suggested in Table 2. We expect this tuning to get harder with increasing FLSPs, as demonstrated by Fig. 1. Through this empirical analysis of the FL algorithms over varying FLSP configurations, we expect to find matching between the optimal algorithm to be used for the given FLSPs. Furthermore, to improve performance measurement, we will understand the various suggested metrics in literature and provide an algorithm-matching based on the right FLM and the given FLSPs. Our experiments will primarily consist of using the three different baseline datasets combined with four prominent FL algorithms in the literature to develop this matching. Precisely, the experiments will be based on:

- the variation in the FLSPs,
- performance measurement over multiple metrics and determining the right FLM,
- computing over a multitude of baseline FL datasets and algorithms.

FLSPs: Prompted from previous literature works (Hu et al., 2020), (Liang et al., 2020), we first list the possible FLSPs in Table 1. Due to the limited scope of this article, we decided to focus on the parameters related to FL solely. We do understand that a lot of supplementary parameters exist for DP, SMPC, and HE. We, however, defer this work to future articles.

Metrics: Several metrics are suggested to keep track of how an FL system performs. We list the proposed metrics in Table 2. We realize that some metrics target the budget available while others define the performance of the algorithm and appropriately segregate them. Therefore, our experiments involve tracking each one of these metrics to understand how the FLSPs and the choice of algorithms affect them. Similar to FLSPs, we postpone

Metrics	**Description**
Model	Accuracy and loss
Fairness	Similar model performance over clients
Communication Cost	Number of global rounds
Computational Power	Number of local rounds

Table 2: FL Metrics

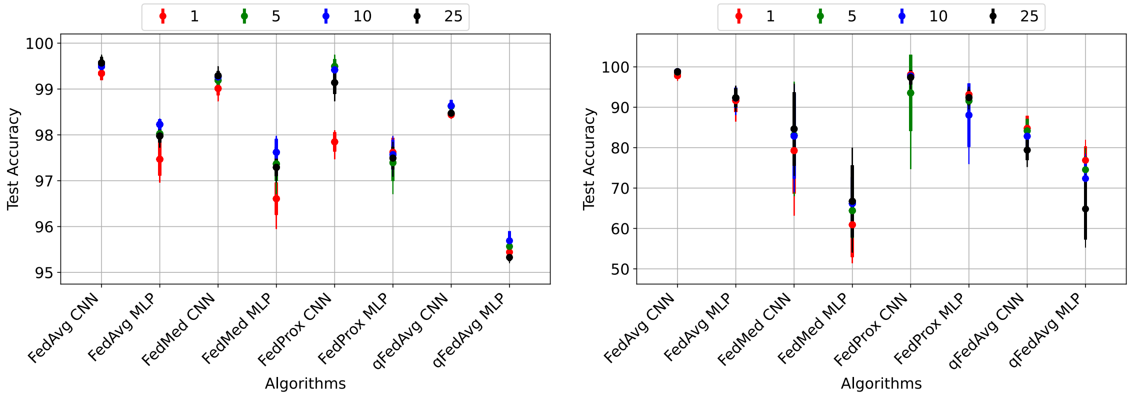
discussing metrics relating to DP, SMPC, and HE to future work.

FL algorithms and datasets: We decide to start with four baseline Federated Algorithms - FedAvg (McMahan et al., 2017), FedProx (Li et al., 2018), FedMed (Pillutla et al., 2019; Yin et al., 2021), and qFedAvg (Li et al., 2019). These cover different optimisation methods, synchronicity, robustness, and fairness in FL.

Embracing Open-Science: Finally, besides the FLSPs, we realize that a number of articles in FL lack implementations of the proposed algorithm, therefore requiring heavy hyperparameter tuning and adjustments to the proposed ML model. We, therefore, will provide implementations of all our code, where new implementations will continually be added.

Parameters	**Description**	**Default Values**
Number of clients (K)	Total clients participating in an FL training	100
Client fraction (C)	Fraction of clients selected for updates in a given round	0.1
Number of Local updates (E)	Number of training epochs of a client	5
Number of global rounds (R)	Number of synchronous aggregation of local updates	50
Number of experimental runs	Number of times the experiments are repeated for each FLSP	5
Batch size (B)	The training batch size at each client	10
Learning rate (lr)	The learning rate constant for a chosen neural network architecture	0.05
Proximal term constant (μ)	For stability in heterogenous settings	0.01
Number of malicious clients (M)	Fraction of clients which are malicious	0.01

Table 3: Various parameters, their description, and default values.



(a) IID Data Distribution

(b) Non-IID Data Distribution

Figure 2: **Number of Local Updates [MNIST]:** In these plots, we demonstrate error graphs for model test accuracies while varying the number of local updates (over $\{1, 5, 10, 25\}$); the left hand side demonstrates results over IID MNIST dataset; the right hand side figure shows the results for the non-IID scenario on the MNIST dataset.

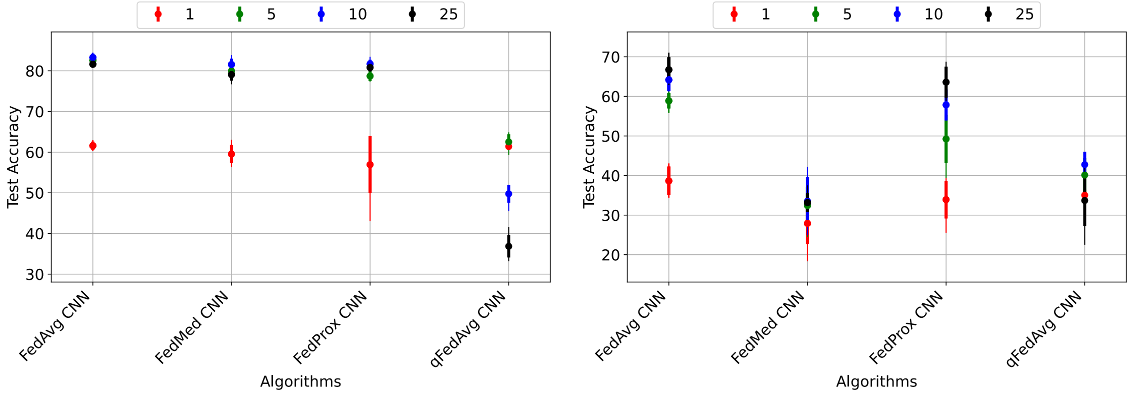
4. Results

In this section, we present the experimental configurations and results for each of the FLSPs reported in Table 1. We vary a chosen FLSP while keeping the remaining ones fixed for the entire runs of experiments. Each experiment is repeated five times, and the average values are compared for each FL algorithm.

In this study, we have used three different datasets – MNIST, CIFAR-10, and Shakespeare. Both MNIST and CIFAR-10 are 10-class datasets with MNIST containing handwritten digits, while CIFAR-10 constitutes images from ships, trucks, birds, and other objects. Shakespeare is a text dataset containing dialogues of 715 users from Shakespeare's work. For the MNIST dataset, we used two different neural network architectures: 1) A multilayer-perceptron with 2-hidden layers with 200 units each using ReLu activations, which we refer to as the MLP. 2) We also use a Convolution Neural Network (CNN) with two 5x5 convolution layers (the first with 32 channels, the second with 64, each followed with 2x2 max pooling), a fully connected layer with 512 units and ReLu activation and a final softmax output layer. We call it MNIST CNN.

In the case of CIFAR, we have used CIFAR CNN[1] – a deeper variant of CNN with three convolution blocks and two fully connected layers followed by a softmax output layer. We skipped the use of MLP for CIFAR as it is too simple to learn a more complex dataset such as CIFAR and thus performs poorly, making it less useful for our experiments. For the Shakespeare dataset, a character-level LSTM language model is used to predict the next character. The neural network architectures used are adapted from McMahan et al. (2017). The default value for the learning rate (lr) in Shakespeare LSTM is set to 0.8.

[1] `https://zhenye-na.github.io/2018/09/28/pytorch-cnn-cifar10.html`



(a) IID Data Distribution

(b) Non-IID Data Distribution

Figure 3: **Number of Local Updates [CIFAR]:** In these plots, we demonstrate error graphs for model test accuracies while varying the number of local updates (over $\{1, 5, 10, 25\}$); the left hand side demonstrates results over IID CIFAR dataset; the right hand side figure shows the results for the non-IID scenario on the CIFAR dataset.

The major parameters used in this paper are briefly explained in Table 3. Wherever not explicitly reported, the default values of each of the parameters are also depicted in the table.

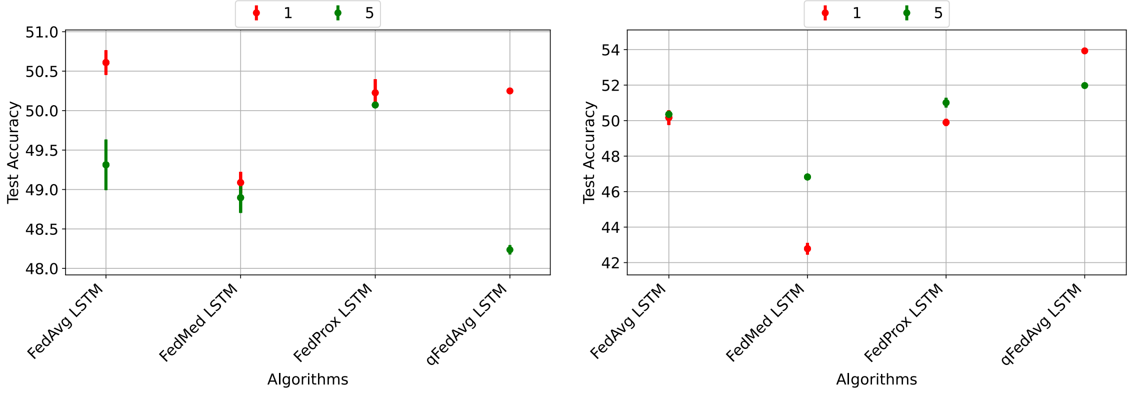
4.1. Number of Local Updates

We investigate the effect of varying the number of local updates on the performance of the trained model. We note that each local update might have a significant impact on client resources, especially in low-resource environments. On the other hand, we expect more local rounds to improve per-client and overall performance.

We define a single pass of training over the client data as one local update of the client. We explore the model performance while we vary the number of local updates (E) as {1, 5, 10, 25} on the MNIST, CIFAR datasets and {1, 5} local updates on the Shakespeare dataset (presented in Figs. 2, 3, and 4, respectively).

In MNIST and CIFAR datasets, we observe that having more than one local update per round is hugely beneficial for FedAvg, FedMed, and FedProx algorithms. A few notable exceptions are the Non-IID MNIST dataset and the qFedAvg algorithm, where the number of local updates does not seem to affect model performance. We further note that increasing the number of local updates beyond five is disadvantageous as it consumes extra resources while maintaining the model performance.

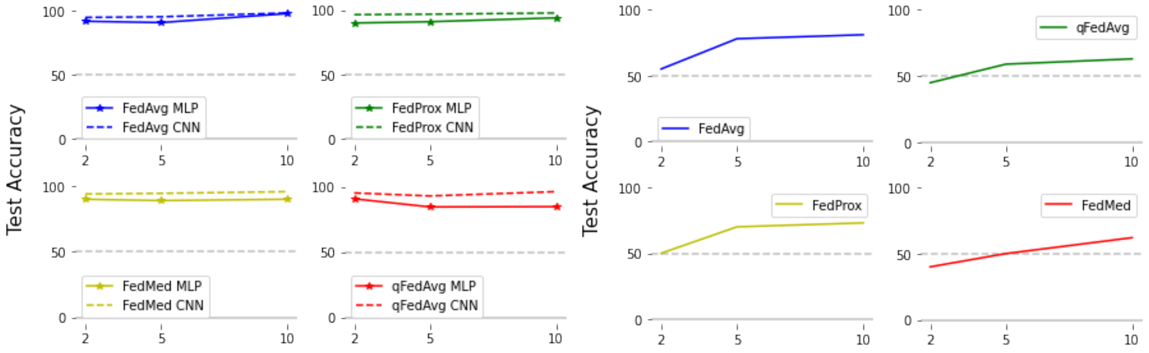
Alternatively, the IID Shakespeare dataset performs inversely, where the model performance for all the algorithms decreases with the increase in local updates. Thus, we recommend fewer client updates per round. For the Non-IID Shakespeare dataset, we find that both FedMed and FedProx benefit with increasing local updates, while FedAvg re-



(a) IID Data Distribution (b) Non-IID Data Distribution

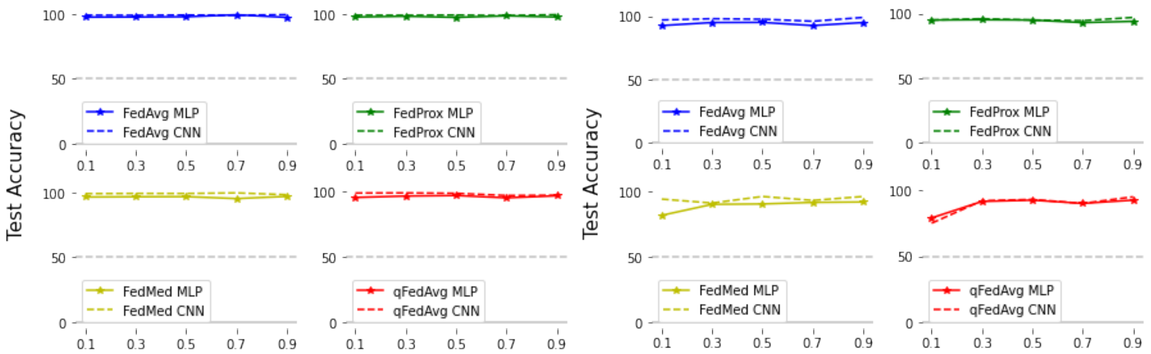
Figure 4: **Number of Local Updates [Shakespeare]:** In these plots, we demonstrate error graphs for model test accuracies while varying the number of local updates (over $\{1, 5\}$); the left hand side demonstrates results over IID Shakespeare dataset; the right hand side figure shows the results for the non-IID scenario on the Shakespeare dataset.

mains constant. As observed in the IID case, qFedAvg performs worse for increasing local updates.



(a) MNIST (b) CIFAR

Figure 5: **Skewness:** This figure demonstrates the effect of varying the degree of skewed class distributions ($\{2, 5, 10\}$) *shards*; on the left hand side are the test accuracy results for the MNIST dataset; on the right hand side we have the results on the CIFAR dataset over varying number of *shards*.



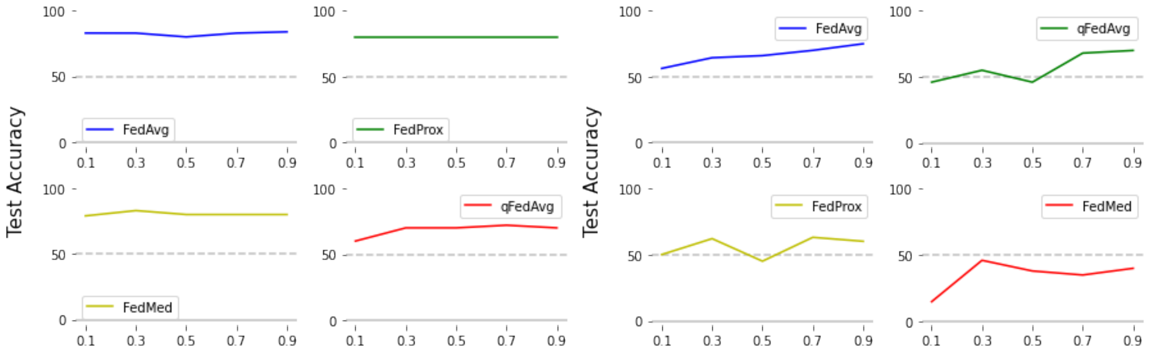
(a) IID Data Distribution

(b) Non-IID Data Distribution

Figure 6: **Client Fraction MNIST:** In this figure, we demonstrate the model performance over variations in the client fraction (over $\{0.1, 0.5, 0.9\}$); the left hand side figure displays the test accuracies of our entire suite of algorithms and architectures for the MNIST dataset distributed in an IID fashion; on the right we have the results for the non-IID setting on the MNIST dataset.

4.2. Skewness and number of clients

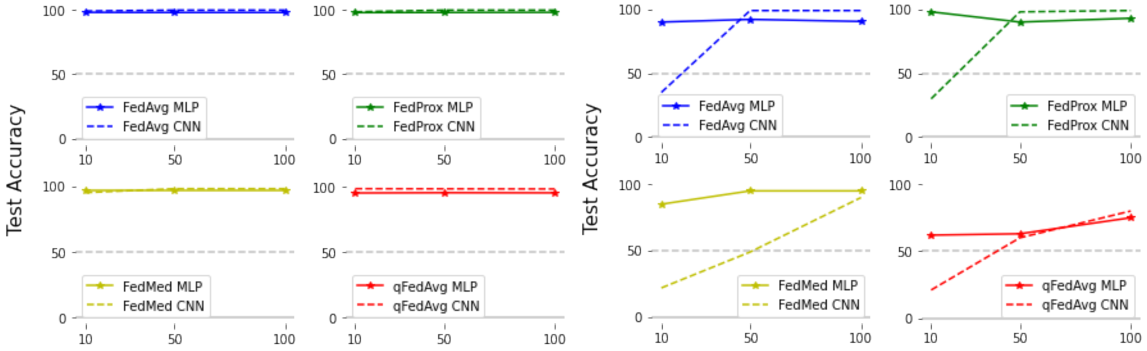
A common reason for the change in model performance is based on the way the data is distributed (*shards*), the number of clients active in each round (K), and the total number of clients (C). In order to better observe major changes in these FLSPs, we use sparkline graphs to be able to capture trends easily. Thus, we have separated out graphs rather than vanilla line graphs.



(a) IID Data Distribution

(b) Non-IID Data Distribution

Figure 7: **Client Fraction CIFAR:** In this figure, we demonstrate the model performance over variations in the client fraction ($\{0.1, 0.3, 0.5, 0.7, 0.9\}$); the left hand side figure displays the test accuracies of our entire suite of algorithms and architectures for the CIFAR dataset distributed in an IID fashion; on the right we have the results for the non-IID setting on the CIFAR dataset.



(a) IID Data Distribution

(b) Non-IID Data Distribution

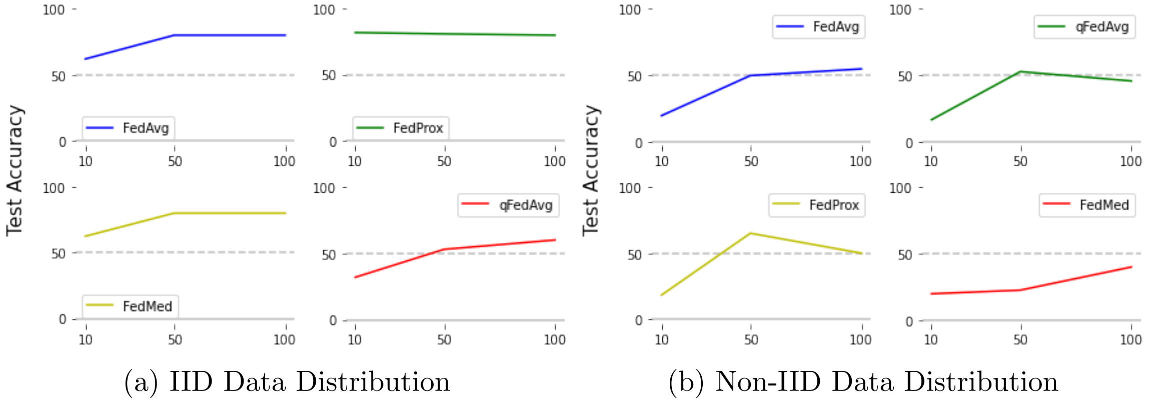
Figure 8: **Number of Clients [MNIST]:** The above figures demonstrate model performance over varying total number of clients ($\{10, 50, 100\}$); left hand side figure displays test accuracy levels on the IID MNIST dataset; the right hand side figure demonstrates accuracies for the non-IID MNIST dataset.

We model the skewness only for the Non-IID case on the MNIST and CIFAR datasets and describe it in terms of *number of shards per user*. A *shard* is the number of classes held by a single user. We vary our skewness over {2, 5, 10} *shards*. In the case of MNIST, this denotes the digit classes held by each user. We can clearly see from Fig. 5(b) that for the CIFAR dataset, by increasing the shards (and thus reducing the skewness) we obtain better overall accuracy. However, for MNIST, the shards do not seem to matter (as shown in 5(a)). One exception is the qFedAvg which actually suffers in the case of MLP architecture with increased shards.

Next, we perform similar variations on the number of clients {10, 50, 100} as well as the client fractions {0.1, 0.3, 0.5, 0.7, 0.9}. We run these two experiments under both IID and Non-IID settings for all the different algorithms using MNIST and CIFAR datasets.

We notice the impact of the client fractions on the model performance as shown in for MNIST and CIFAR in Figs. 6 and 7 respectively. Here the client fraction denotes the fraction of clients chosen for an update from the total clients. We can clearly see that for almost all the algorithms, the increase in client fractions seems to improve the performance in the Non-IID setting. In comparison for IID datasets, there seems to be no clear trend for any of the algorithms. In fact, often we see similar performance for all the client fractions. A notable exception includes the qFedAvg algorithm, which significantly improves on the IID CIFAR dataset.

Similarly, the total number of clients tends to increase the overall performance. We present these for the MNIST and CIFAR in Figs. 8 and 9 respectively. Specifically, we see that even using half of the total clients (50) seems to provide a significant and sometimes even a globally optimal performance. FedMed and qFedAvg provide boosts in certain scenarios for 50+ clients. A good recommendation is to train with the maximum possible clients to improve the model since performance rarely degrades with an increase in clients (and the subsequent increase in training data points).



(a) IID Data Distribution

(b) Non-IID Data Distribution

Figure 9: **Number of Clients [CIFAR]:** The above figures demonstrate model performance over varying total number of clients (over $\{10, 50, 100\}$); left hand side figure displays test accuracy levels on the IID CIFAR dataset; the right hand side figure demonstrates accuracies for the non-IID CIFAR dataset.

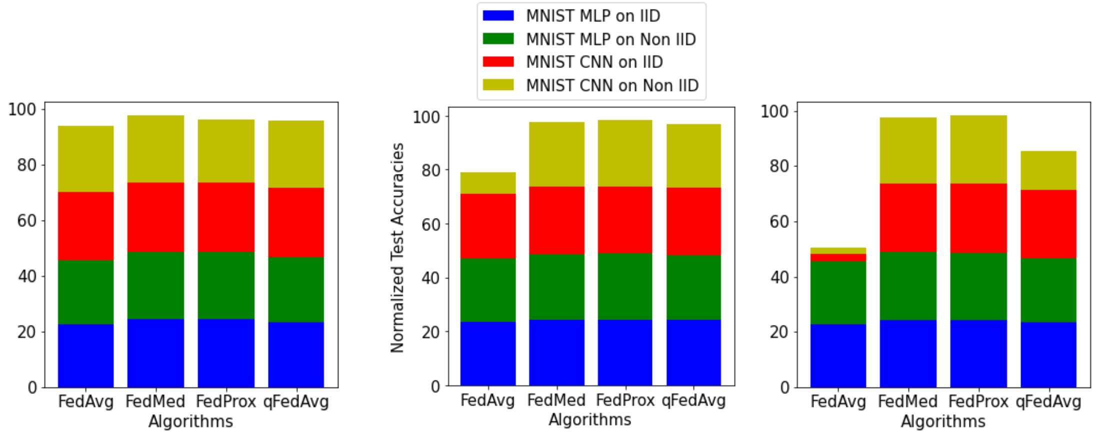


Figure 10: **Effect of Stragglers on MNIST dataset:** From left to right, the number of straggler are set to 10%, 50%, 90% respectively.

4.3. Effect of Stragglers

Heterogeneity is a significant challenge in any communication-oriented system, such as in the case of FL. In a real-world scenario, not all the clients are able to provide their local updates in synchronization with the other clients. These irregular clients (slow clients) are termed as *stragglers* in an FL setting (Li et al., 2018).

To simulate the stragglers, we choose a certain fraction of clients from the selected clients (C) to act as *slow workers* and completely discard their updates for that particular round. The effect of stragglers for different slow fraction (S) values (10%, 50%, 90% i.e. $S = 0.1$,

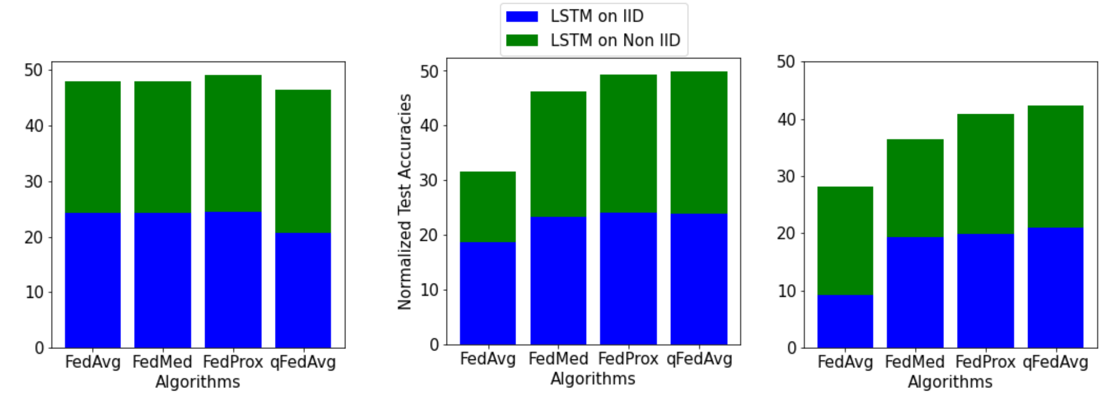


Figure 11: **Effect of Stragglers on Shakespeare dataset:** From left to right, the number of straggler are set to 10%, 50%, 90% respectively.

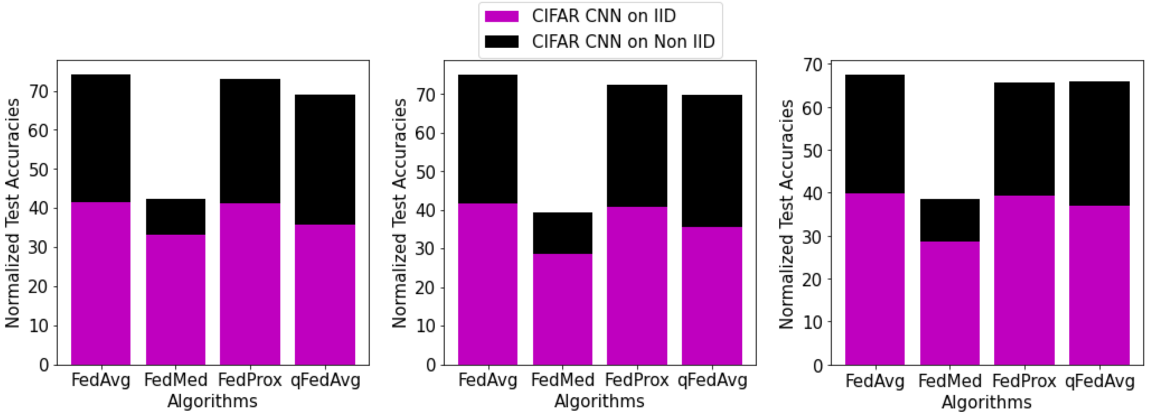
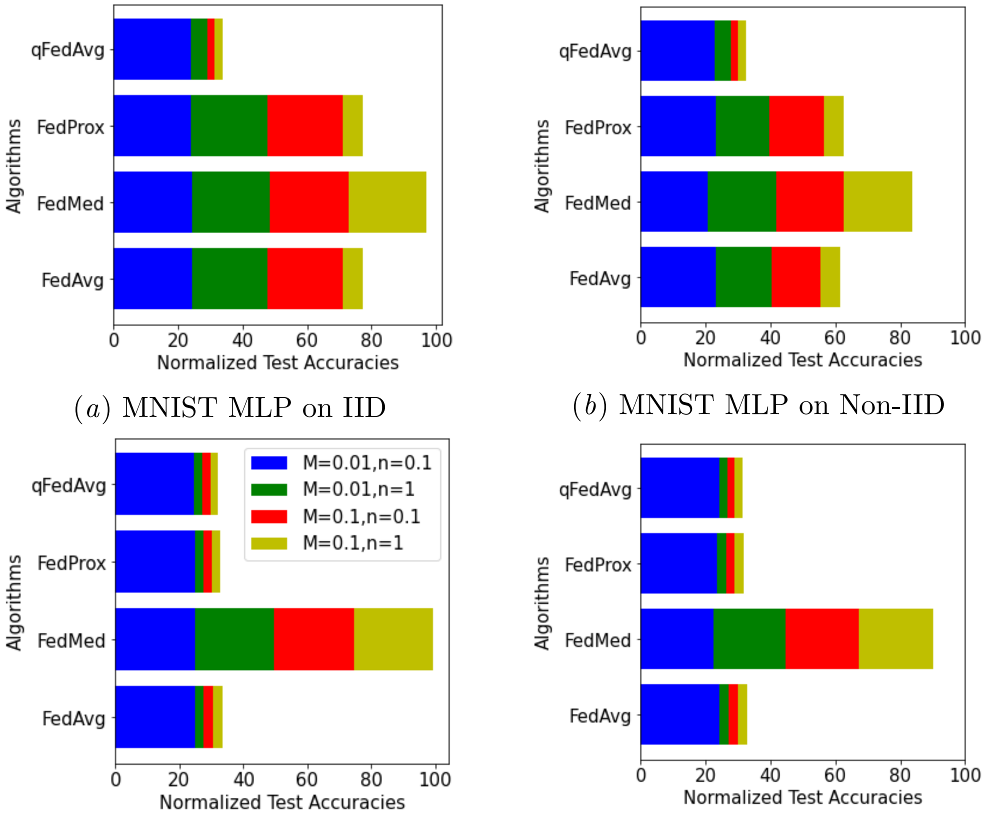


Figure 12: **Effect of Stragglers on CIFAR-10 dataset:** From left to right, the number of straggler are set to 10%, 50%, 90% respectively.

$S = 0.5$, $S = 0.9$) on MNIST, CIFAR, and Shakespeare datasets are presented in Figs. 10, 11, and 12, respectively.

The test accuracies for the MNIST dataset is normalized by a factor of four (as we portray results of two different neural networks; an MLP and a CNN in both IID and Non-IID settings), while for the CIFAR and Shakespeare experiments, the normalization is done by a factor of two since we evaluate these datasets on the same neural network architecture. As can be seen from Figs. 10 and 11, the accuracy of FedAvg drastically decreases as the fraction of stragglers (S) is increased in both MNIST and Shakespeare datasets (which also aligns with the results from Li et al. (2018)). However, in the case of the CIFAR dataset (Fig. 12), FedMed takes a significant drop in the accuracy compared to other algorithms. The results show evidence that both FedAvg and FedMed are a poor candidate for FL scenarios involving high heterogeneity and stragglers.



(a) MNIST MLP on IID

(b) MNIST MLP on Non-IID

(c) MNIST CNN on IID

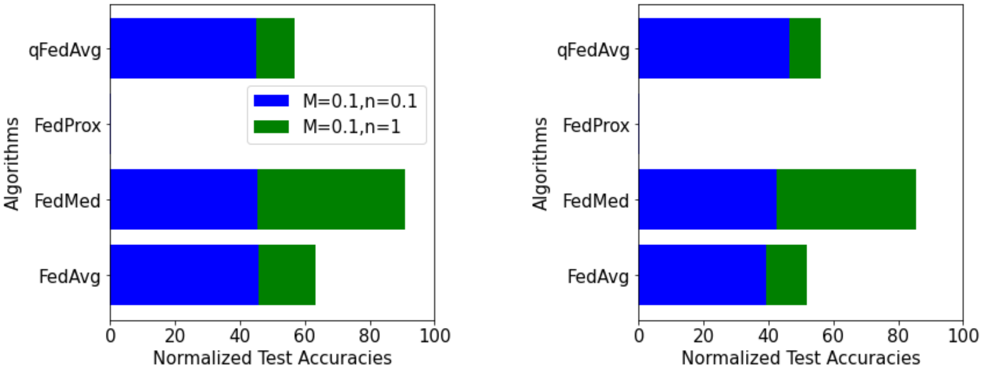
(d) MNIST CNN on Non-IID

Figure 13: **Robustness [MNIST]:** Effect of local model poisoning attacks on the accuracy for MNIST dataset.

4.4. Robustness

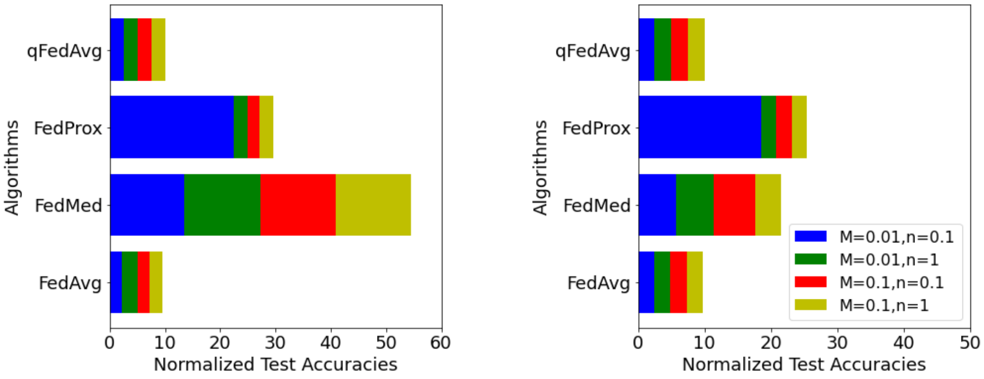
An FL training session incorporates updates from multiple distributed clients. It is desirable that an FL algorithm is robust to corrupt updates from the clients. A corrupt update could be caused by a malfunctioning hardware, a software bug, or due to the presence of adversaries. In the scope of this paper, we intend to evaluate the robustness of our four chosen FL algorithms against varied conditions. In our experiments, we employ local model poisoning attacks and take inspiration from Fang et al. (2020). We assign active malicious clients (M) with a given noise element (n). This means, each update by a malicious client is sent to the server as a vector of model parameter values (or weights) equal to $[n, n, ..., n]$. We chose the constant parameters as: $C = 1$, $K = 100$, $E = 5$, $B = 10$ $\mu = 0.01$, $lr = 0.05$, $q = 0.001$, and number of global rounds, $R = 10$.

The results obtained from varying the number of malicious clients (M) and noise element (n) for MNIST, Shakespeare, and CIFAR datasets are show in Figs. 13, 14, and 15



(a) Shakespeare LSTM on IID (b) Shakespeare LSTM on Non-IID

Figure 14: **Robustness [Shakespeare]:** Effect of local model poisoning attacks on the accuracy for Shakespeare dataset.



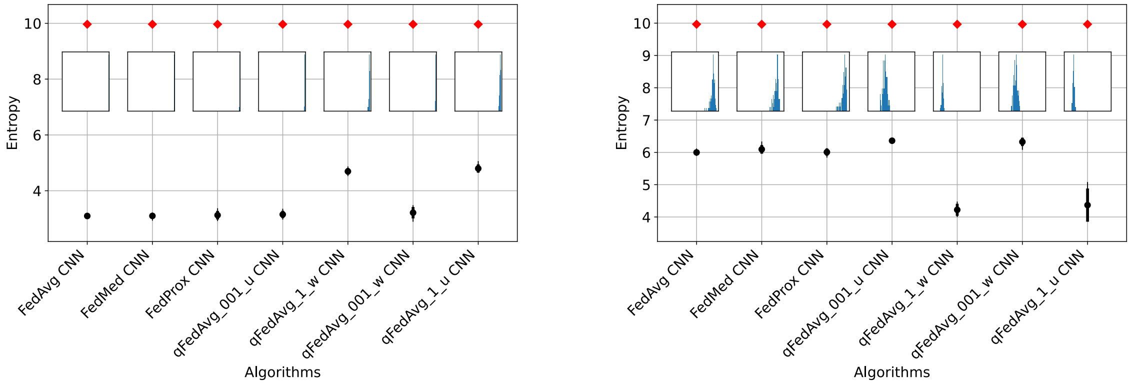
(a) CIFAR CNN on IID (b) CIFAR CNN on Non-IID

Figure 15: **Robustness [CIFAR]:** Effect of local model poisoning attacks on the accuracy for CIFAR-10 dataset.

respectively. We consider four settings where for each $M = 0.01$ and $M = 0.1$ we initialise noise element as $n = 0.1$ or $n = 1$. As can be observed from the figures, FedMed algorithm outperforms other algorithms for different pairs of M and n values.

4.5. Fairness

Fairness in FL can have multiple notions. Fairness can be defined as unbiased behavior towards protected classes such as race and gender. However, in this paper, we define fairness with respect to client performances (Li et al., 2018). More concretely, we call an algorithm more fair if the individual test accuracies of clients are closer to each other. We



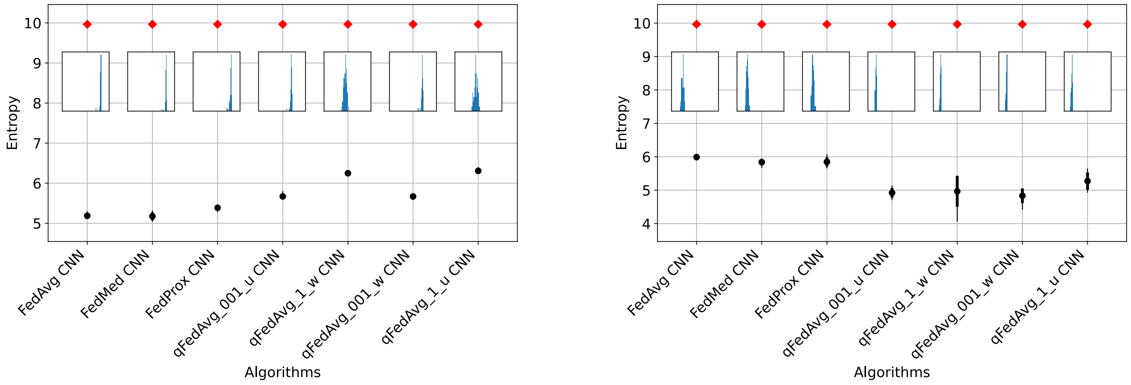
(a) IID Data Distribution

(b) Non-IID Data Distribution

Figure 16: **Evaluating fairness [MNIST]:** We have used entropy as an indication of fairness where lower values of entropy for an algorithm signifying better fairness. The accuracy histograms for all the clients using each algorithm is superimposed on the entropy graph.

present a method to quantify the fairness of the algorithm using Shannon entropy (in bits) (Shannon, 1948). We train the clients using a chosen FL algorithm and create the histogram of client test accuracies. We normalize this histogram to convert it into a valid probability mass function (*pmf*) and calculate its entropy. Ideally, a perfect fair algorithm will have a histogram with all the clients binned to the highest accuracy bin. This will translate to a *pmf* with value one at the highest accuracy bin and zero elsewhere. The entropy of this ideal distribution can be easily observed to be zero. We can generalize this observation to the practical case by considering the algorithms which have lower entropy of the *pmf* to be more fair. We choose to have 1000 accuracy bins, and hence, the worst fair algorithm would have clients uniformly distributed across all bins. We calculate the entropy of the worst fair algorithm and depict it in the plots as red diamond symbols, i.e., the lower the entropy value for an algorithm, more fair it is to the clients' performance.

We conduct additional experiments for the qFedAvg algorithm since it is specifically crafted for fair resource allocation. We evaluate the performance of qFedAvg for uniform and weighted client sampling (labeled as u and w in the plots, respectively). Uniform client sampling refers to weighting devices equally when sampling devices and weighted client sampling refers to weighting devices adversarially to optimize the worst-performing devices. We also evaluate the performance of qFedAvg for q values 0.001 and 1 (labelled as *001* and *1* in the plots respectively). The parameter q controls the amount of fairness that should be imposed while training. A larger value of q will give magnify the losses of the clients with larger local losses. This enhances the uniformity, and thus the fairness, of the model performances across clients. The definition of sampling methods and q follows from (Li et al., 2018).

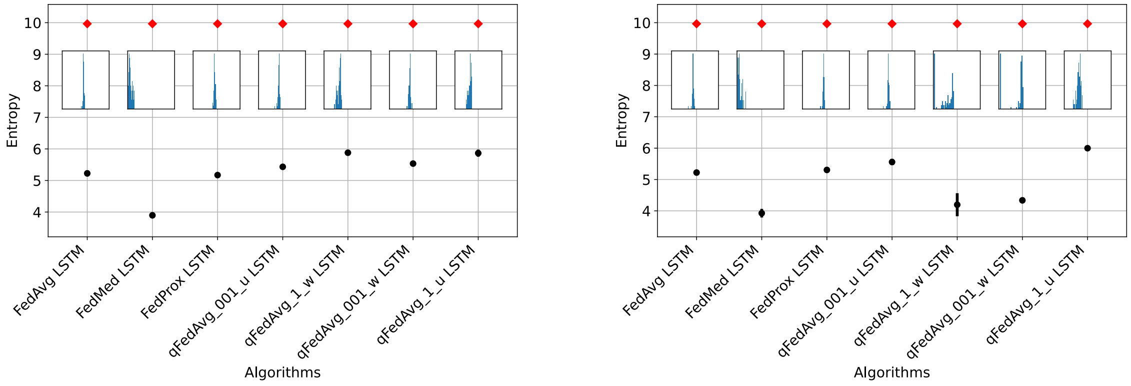


(*a*) IID Data Distribution

(*b*) Non-IID Data Distribution

Figure 17: **Evaluating fairness [CIFAR]:** We have used entropy as an indication of fairness where lower values of entropy for an algorithm signifying better fairness. The accuracy histograms for all the clients using each algorithm is superimposed on the entropy graph.

To condense the information, we superimpose the client performance histograms on the entropy graph. Figs. 16, 17, and 18 presents the results for MNIST, CIFAR, and Shakespeare datasets respectively. We can observe that qFedAvg is less fair in the IID



(*a*) IID Data Distribution

(*b*) Non-IID Data Distribution

Figure 18: **Evaluating fairness [Shakespeare]:** We have used entropy as an indication of fairness where lower values of entropy for an algorithm signifying better fairness. The accuracy histograms for all the clients using each algorithm is superimposed on the entropy graph.

setting and is more fair in the Non-IID setting compared to FedAvg, FedProx, and FedMed with all the datasets. Except for the Non-IID Shakesphere dataset, the client sampling methods (*uniform*, *weighted*) in qFedAvg does not have a noticeable impact on fairness. Contrary to what we would expect, we find a larger q value in qFedAvg for IID datasets to decrease the fairness. This suggests that the parameter q will need careful tuning for better fairness in IID settings. For the Non-IID MNIST dataset (Fig. 16(*b*)), we can observe that a larger q value increases the fairness regardless of the sampling methods. qFedAvg seems to be independent of the sampling methods, and the q value for the Non-IID CIFAR dataset as the same fairness is achieved in all cases. Interestingly, for the Non-IID Shakespeare dataset, the weighted sampling scheme outperforms the uniform sampling scheme regardless of the q value. The histograms for the IID MNIST dataset, which is probably the simplest case, are extremely skewed and concentrated towards the right end of the histogram (owing to high accuracies).

5. Findings

From our results we see that having more than one local update is useful in most scenarios. Especially, algorithms FedAvg, FedMed and FedProx seem to benefit from this approach and perform equally well on most datasets (MNIST and CIFAR). However, it turns out that on Shakespeare dataset less is in fact more and fewer local updates perform better. Further qFedAvg performs close to optimal for either data distribution.

Reducing skewness improves accuracy for the CIFAR datasets across algorithms. The FedAvg and qFedAvg perform the best for CIFAR under skewness. As expected FedMed performs the worst since it is not able to utilize the increased diversity to the full extent. On MNIST except for qFedAvg all the other algorithms perform similarly. Increased diversity in fact reduces the qFedAvg's performance slightly.

Number of clients drastically improve accuracies of the algorithms. All the algorithms perform well under increasing number of clients except for qFedAvg. Thus, we are inclined to believe that fairness restrictions in qFedAvg is the only cause for a dip in performance. Therefore, for overall performance we will tend to use other algorithms. Client Fraction exhibits a similar increasing trend especially for Non IID datasets.

In the case of stragglers, we observe that FedMed performs poorly for CIFAR. This is due to the fact that FedMed computes a geometric median of the client updates which many a times will not be able to merge the information from the client updates. As can be seen from Figs. 10, 11, and 12, both FedProx and qFedAvg performs comparable. However, we suggest that FedProx would be a suitable choice for straggler settings as it requires lesser parameters to tune and it also derives its implementation from the most commonly used FL architecture, FedAvg. With low number of stragglers, FedProx will be better than qFedAvg since there is a performance trade-off for fairness in the case of qFedAvg.

FedMed is a clear winner when the client updates are noisy which makes it more robust. However, it is to be noted that we tested our hypothesis on local model poisoning attacks where FedMed outperforms other algorithms. This observation may not be true for other forms of attacks.

Except qFedAvg, all the other algorithms tend to optimise the performance of the global model without caring about the local performance at a client. Thus, one can clearly see

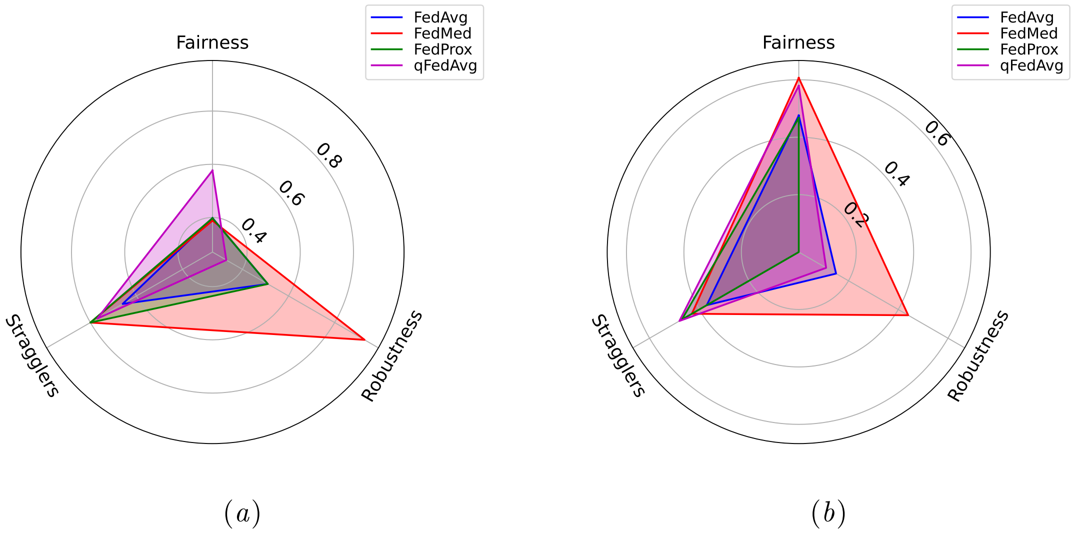


Figure 19: Spider graphs for (a) MNIST and (b) Shakespeare datasets.

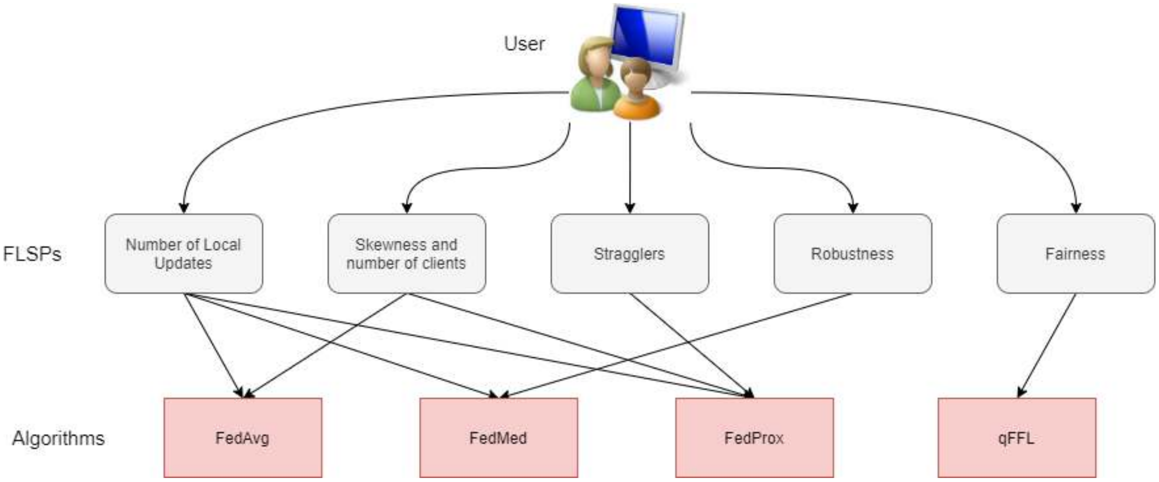


Figure 20: Analytical Hierarchy Process (AHP) with user at top level, FLSPs in the mid-level and Algorithms as leaf nodes. The user can select algorithms based on his/her FLSP requirements.

high entropy values for FedAvg, FedProx, and FedMed in Figs. 16, 17, and 18 which denotes a lower fairness. A q value of 1 for qFedAvg algorithm provides a better fairness than the other q values.

To conclude, the spider graphs presented in Fig. 19 represent how each algorithm fares based on the FLSPs evaluated in this paper. For better representation, we convert the fairness measure to (1 - ($^{entropy}/_{max\_entropy}$)). In Fig. 19($a$), we can observe that qFedAvg performs better than other algorithms with respect to Fairness, while for Robustness, FedMed does well. In the case of stragglers, FedMed, qFedAvg, and FedProx closely contest but we recommend FedProx for maintaining system heterogeneity. While in Fig. 19(*b*), FedMed is an overall better choice than other algorithms forthe Shakespeare dataset.

Furthermore, Fig. 19 allows us to to scale the number of metrics and algorithms visually represented easily. In scenarios where we wish to make decisions based on joint optimality of metrics, we can easily obtain a weighted spider graph based on the importance of each metric to make quicker algorithm decisions. Note that we are only required to create the graphs for our custom FLSP setting. Moreover, we only need to include the FLSPs which are significantly dependent on the algorithm choice.

Fig. 20 display the Analytical Hierarchy Process (AHP) (Taherdoost, 2017) on how a user can make a decision on which algorithm to choose based on his/her requirements. In this AHP, the arrows from the FLSPs point to the algorithms which provide the best performance. For future work, we would like to add the ability to provide weighted decisions over the FLSPs. We plan to release the code for all the experiments carried out in this paper. We hope that more benchmarks can be added in the future by the open-source contributors, which will help in expanding our work to upcoming FL algorithms.

References

Mohammed Aledhari, Rehma Razzak, Reza M Parizi, and Fahad Saeed. Federated learning: A survey on enabling technologies, protocols, and applications. *IEEE Access*, 8:140699–140725, 2020.

George J. Annas. Hipaa regulations — a new era of medical-record privacy? *New England Journal of Medicine*, 348(15):1486–1490, 2003. doi: 10.1056/NEJMlim035027.

Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to byzantine-robust federated learning. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1605–1622. USENIX Association, August 2020. ISBN 978-1-939133-17-5. URL https://www.usenix.org/conference/usenixsecurity20/presentation/fang.

Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

Sixu Hu, Yuan Li, Xu Liu, Qinbin Li, Zhaomin Wu, and Bingsheng He. The oarf benchmark suite: Characterization and implications for federated learning systems. *arXiv preprint arXiv:2006.07856*, 2020.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. *arXiv preprint arXiv:1905.10497*, 2019.

Yuan Liang, Yange Guo, Yanxia Gong, Chunjie Luo, Jianfeng Zhan, and Yunyou Huang. An isolated data island benchmark suite for federated learning. *arXiv preprint arXiv:2008.07257*, 2020.

Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3): 2031–2063, 2020.

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

Adrian Nilsson, Simon Smith, Gregor Ulm, Emil Gustavsson, and Mats Jirstrand. A performance evaluation of federated learning algorithms. In *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, pages 1–8, 2018.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. Robust aggregation for federated learning. *arXiv preprint arXiv:1912.13445*, 2019.

Tushar Semwal, Promod Yenigalla, Gaurav Mathur, and Shivashankar B Nair. A practitioners' guide to transfer learning for text classification using convolutional neural networks. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 513–521. SIAM, 2018.

Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Hamed Taherdoost. Decision making using the analytic hierarchy process (ahp); a step by step approach. *International Journal of Economics and Management Systems*, 01 2017.

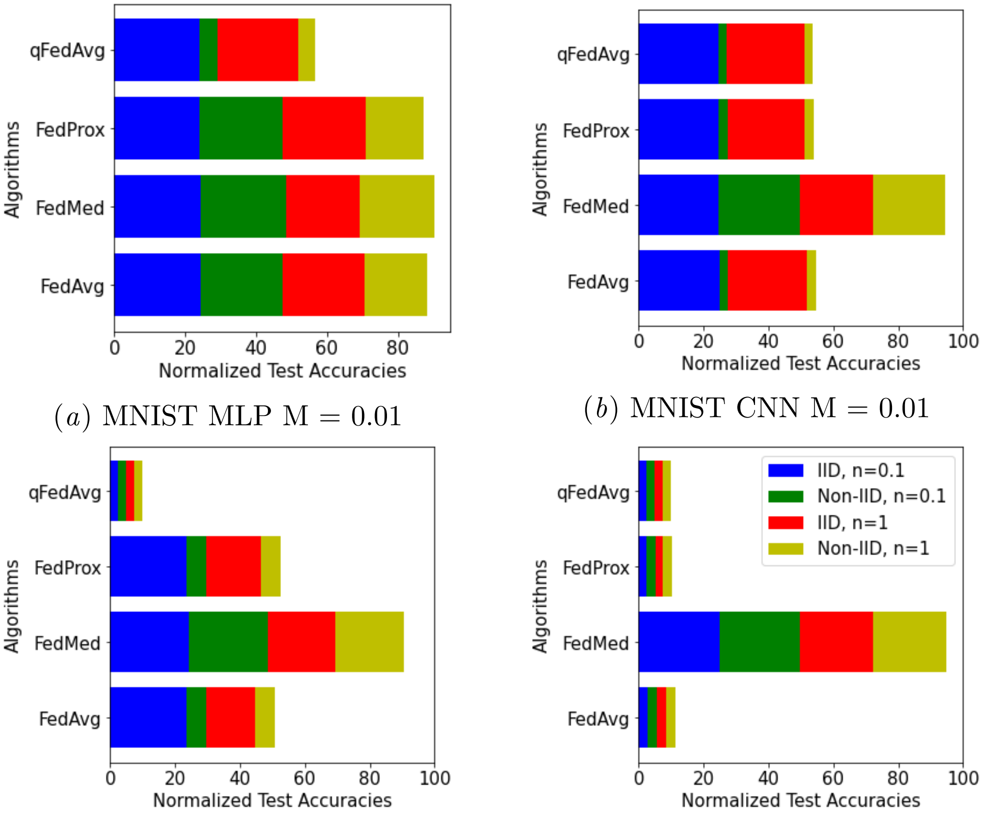
Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. 2021.

Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 253–263, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing.

Appendix A.

Figs. 21(*a*), 21(*b*), 21(*c*), 21(*d*) refer to how robustness could be classified against the varying level of adversaries (malicious clients) in the protocol.



(*a*) MNIST MLP M = 0.01

(*b*) MNIST CNN M = 0.01

(*c*) MNIST MLP M = 0.1

(*d*) MNIST CNN M = 0.1

Figure 21: Comparing robustness of the algorithms based on the paired values of data distribution and noise element for MNIST dataset. This graph is different from Fig.16 since it compares across the distribution of data and thus provides a different perspective.