A Survey of Federated Learning

Adira Cohen, Qiurui Du, Miles Woollacott

ST758, Spring 2025

Abstract

In this report, we introduce Federated Learning (FL). When handling decentralized privacy-protected data, FL can be used to fit machine learning models without sharing data between the owners of the datasets. FL was first formalized in 2017 by McMahan et al. [1], and has since expanded to a wide array of applications. We will review several recent applications in this report to give a flavor of the types of modifications that can be made to suite different use-cases. First, we will discuss Low-Parameter Federated Learning, introduced by Jiang et al. [2], which aims to make Large Language Models tractable to a FL setting. Next, we will discuss the FedFed algorithm, introduced by Yang et al. [3], which seeks to combine the best aspects of FL and more standard centralized learning. Finally, we will discuss a poisoning-attack protection model developed by Yazdinejad et al. ([4]) which uses a series of security and optimization techniques to preserve data privacy from malicious actors without sacrificing computational efficiency. We also conduct a comprehensive numerical study in the field of computer vision to evaluate the performance, sensitivity, and robustness of two mainstream FL frameworks: FedAvg and FedProx. Results show that FedAvg, as a pioneering method in FL, performs best under ideal, non-heterogeneous settings, achieving strong performance without compromising data privacy, and showing low sensitivity to hyperparameters, making it broadly applicable. However, in the presence of heterogeneity, where FL often leads to performance degradation compared to centralized models, FedProx demonstrates greater robustness and becomes the preferable choice.

1 Introduction

In this report, we provide a survey of the machine learning (ML) framework called Federated Learning (FL). When our data originates from multiple sources (hence referred to as parties/clients/devices) where data privacy is an issue, we must keep the data decentralized. In addition, this data cannot be assumed to be independent and identically distributed (i.i.d.). Therefore, we must introduce a new framework that can tackle such situations.

Federated Learning was first introduced in a 2017 paper via the FederatedAveraging (FedAvg) algorithm [1] to train Large Language Models (LLMs) on mobile devices, detailed in Figure 1. Generally speaking, FL executes a call-and-response between a central server (aggregator) and all participating parties for a pre-determined number of rounds. At each round, the central server sends parameters that the parties then train an ML model with on their individual datasets, and parties send back updates to the central server. The central server then aggregates these updates in some fair way [5]. Applying this to FedAvg, the algorithm has each mobile device train an stochastic gradient descent (SGD) model (thus reporting back gradient vectors as updates), which then the central server aggregates via a weighted average based on the availability of data in each device. While FedAvg is a valuable starting point for FL as a whole, it will be shown in section 3 that it does not perform well under a non-i.i.d. setting.

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
$m \leftarrow \max(C \cdot K, 1)$
$S_t \leftarrow$ (random set of m clients)
for each client $k \in S_t$ **in parallel do**
$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
for batch $b \in \mathcal{B}$ **do**
$w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

Figure 1: *FedAvg* algorithm introduced in [1]. This is the algorithm that introduced FL.

FL excels at upholding data privacy [5]. The data remains decentralized, and only updates are communicated between the parties and central server. This means that FL has strong applications to areas where data security and privacy are an issue. For instance, FedAvg was designed to keep mobile device data secure. In addition, federal laws like HIPAA prevent the aggregation of medical data [6], and so a natural use case of FL arises with hospitals being the parties in this circumstance.

The structure of the rest of the report is as follows. In section 2, we provide a comprehensive literature review that investigates recent topics of interest in the FL research community. In section 3, we include several simulation studies that demonstrate the capabilities and shortcomings of several FL algorithms.

2 Literature review

In this section, we will discuss three novel techniques to address a selection of common issues in FL.

2.1 LP-FL [2]

Large-language models (LLMs) are useful for improving usability of mobile devices. Often this is in the context of semi-supervised learning. For example, a researcher may want to build a model to predict the sentiment of

social media posts. Some inputs have obvious sentiment labels, such as an upvote or downvote, while others are more difficult to label, such as a product review. This setting suggests the use of FL because the data is often privacy-protected and each individual will have their own typing style and opinions, creating heterogeneous datasets. However, LLMs are too expensive to train with standard FL methods using phones because of the extremely large number of parameters. Another challenge is that most of the phone data is unlabeled.

The general idea of the Low-Parameter Federated Learning (LP-FL) method is to only train a very small subset of the parameters at once and label a small amount of the data at a time. During the client update, the client first assigns labels to some of its unlabeled data. Note that not all of these labels are necessarily correct. Then, the LLM is trained using Low-Rank Adaptation (LoRA) [7] for parameter fine-tuning. LoRA freezes all but a very small subset – sometimes even below 0.25% – of the parameters to greatly reduce the effective trainable size of the model. The pseudocode for LPFL is given in figure 2.

Say we have a LLM M with vocabulary size V and mask token [MASK] $\in V$. Mask tokens are used in LLMs to hide words so that the model must learn to guess the original word using its context. We want to train this LLM using FL, for which we have K clients. Each client $k = 1, \ldots, K$ has a local dataset which is split into a small labeled set T_k and much larger unlabeled set U_k. We give the LLM $P(x)$, which gives the task description $P \in \mathcal{P}$ including a mask token for a given sequence of words x. Then the LLM tries to predict the word in the mask position and associates the given label $l \in L$ with the chosen word $v \in V$. This process is used to predict the probability of each word $v \in V$ being the word masked by the mask token. The score of input x with label l is

$$s_P(l|x) = \sum_{v \in V} M(v|P(x)).$$

The score function is run through the softmax function, which exponentiates the score and then normalizes it. Then the standard cross-entropy loss is given by

$$L_{CE} = -\frac{1}{n} \sum_{i=1}^{n} \log \left(\frac{\exp(s_P(l|x))}{\sum_{l' \in L} \exp(s_P(l'|x))} \right).$$

The cross-entropy loss is used to fine-tune the LLM. To reduce the number of parameters trained, LoRA maps the input dimensions d to r then from r to output size k where $r << \min(d, k)$. More specifically, take original parameters $W_0 \in \mathbb{R}^{d \times k}$ and update with $\Delta W = BA$ where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$. In the local training step, W_0 remains fixed while A and B are trained. After local training, ΔW is sent to the global model and averaged using FedAvg. For loss function f_i for client i and n total clients, this aggregation step is given by

$$\min_{\Delta W} \sum_{k=1}^{K} \frac{n_k}{n} \left(\frac{1}{n_k} \sum_{i \in P_k} f_i(\Delta W) \right).$$

Recall that initially the fraction of the data that is labeled is small. As the global model trains, it is used to assign soft labels to the unlabeled portion U so that they move to the labeled portion T. The prediction accuracy a_P is used as a weight when assigning soft labels s to unlabeled data $x \in U$:

$$s(l|x) = \left[\sum_{P \in \mathcal{P}} a_P \right]^{-1} \sum_{P \in \mathcal{P}} a_P \cdot s_P(l|x)$$

In this way the model is successively trained on an increasing number of labels using collaboration between the local and global models.

There are tradeoffs to the low parameter approach, namely that it decreases potential learning but avoids overfitting. However, the risk seems to pay off, as LP-FL performs very well on classic test datasets, only slightly worse than centralized training. In addition, LP-FL even outperformed not only the full-parameter version but also the centralized version of the model in some settings. Jiang et al. reason that the centralized version is training on a larger number of incorrect labelings at once compared to local training, so reducing the number of parameters reduces overfitting to incorrect labelings. Similarly, the full parameter version can remember noise, i.e. overfit to incorrect labelings.

2.2 FedFed [3]

Often when training a FL model, for example with medical imaging data, there are a small number of features in the data which are much more influential to model performance than the others. This is especially important in a FL setting, where important features may be heterogeneous across sites and therefore the decentralized nature

Server executes:
Initialize global LLM M with LoRA bottleneck modules, and only the parameters of LoRA bottleneck modules w can be trained;
$l \leftarrow 5 \times 10^{-5}$; $G \leftarrow 5$; $E \leftarrow 5$;
while $g \leq G$ **do**
  for each client k *in* K *clients* **in parallel do**
    $w_{g+1}^k \leftarrow$ ClientUpdate (k, w_g)
  $w_{g+1} \leftarrow FedAvg(w_{g+1}^{1:K})$
  Server sends w_{t+1} back to clients
  $g \leftarrow g + 1$
Return M

ClientUpdate(k, w): // *Run on client* k
T is the local labeled data, U is the local unlabeled data
if $g < 5$ **then**
  $S \leftarrow$ label the 25% of the U
  $T \leftarrow T + S$
  $U \leftarrow U - S$
else
  continue
$\mathcal{B} \leftarrow$ (split T and U into batches of size B)
for each local epoch i from 1 to E **do**
  for batch b $\in \mathcal{B}$ **do**
    $w \leftarrow w$ - $\eta \nabla \ell(w; b)$
Return w

Figure 2: LP-FL algorithm with step size l, G global rounds, and E epochs [2]. Note that the novelty is in the client update.

of the model is very disadvantageous. However, raw data cannot be aggregated in the central server because it would violate the data's privacy protections.

The FedFed algorithm addresses this issue by first identifying these important features and then sharing them globally in a protected manner. The data is split data two parts at the client level: performance-sensitive features are the ones critical to model performance which we would like to share globally, and the remaining features we call performance-robust. The performance-sensitive features are selected by an optimization algorithm which seeks to select the smallest subset possible which gives the greatest improvement to model performance. Then noise is added to the performance-sensitive features to preserve privacy. In practice, often very little noise is needed. Finally, these features can be securely shared globally. In this way the FedFed algorithm attempts to approximate a centralized model in a privacy-protected setting. The pseudocode for feature distillation is given in figure 3 and the pseudocode for FL model training is given in figure 4.

Yang et al. specify the distillation of minimal sufficient information Z from input X with label Y as

$$\min_Z I(X; Y|Z), \text{ s.t. } I(X; Z \leq I_{IB})$$

where I is the mutual information and I_{IB} is some constant. In other words, we are minimizing the information between X and Y if we are given Z while ensuring that X and Z can only have a small amount of mutual information.

They then derive a practical objective function, where the goal is to distill minimal performance-sensitive features $z(x; \theta)$ while forcing the preserved features $x - z(x; \theta)$ to be similar to the raw features x. Say we train a local label classifier $f(\cdot; w_k)$ where w_k are parameters for client k, and have cross-entropy loss ℓ. Then the objective function is

$$\min_{\theta, w_k} \left\{ -\mathbb{E}_{(x,y)\sim P(X_k, Y_k)} \left[\ell(f(z(s; \theta; w_k), y) \right], \text{ s.t. } ||z(s; \theta)||_2^2 \right\} \leq \rho$$

for some hyperparameter ρ. We take $z(x; \theta)$ as performance-sensitive features to be shared globally and $x - z(x; \theta)$ as performance-robust features to be stored locally. This objective function is then adjusted by adding noise to the performance-sensitive features using $h(X_k) \coloneqq z(X_k) + n$ with noise n, e.g. Gaussian or Laplacian. Say now that $f(\cdot; \phi_k)$ is trained for model aggregation and (x_p, y) are protected performance-sensitive features. Then the objective function is

$$\min_{\phi_k} \left\{ -\mathbb{E}_{(x,y)\sim P(X_k, Y_k)} \left[\ell(f(x; \phi_k), y) \right] + \mathbb{E}_{(x_p, y)\sim P(h(X_p), Y_k)} \left[\ell(f(x_p; \phi_k), y) \right] \right\}.$$

Algorithm 1 Feature Distillation

Server input: communication round T_d, DP noise level σ_s^2
Client k's input: local epochs of feature distillation E_d, k-th local dataset $\mathcal{D}^k$ and rescale to $[0, 1]$
Initialization: server distributes the initial model $\mathbf{w}^0, \theta^0$ to all clients
Server Executes:
for each round $t = 1, 2, \cdots, T_d$ **do**
  server samples a subset of clients $C_t \subseteq \{1, ..., K\}$,
  server **communicates** $\mathbf{w}^t, \theta^t$ to selected clients
  for each client $k \in C_t$ **in parallel do**
    $\mathbf{w}_k^{t+1}, \theta_k^{t+1} \leftarrow$ Local_FeatDis $(\mathbf{w}^t, \theta^t, \sigma_s^2)$
  end for
  $\mathbf{w}^{t+1}, \theta^{t+1} \leftarrow$ AGG $(\mathbf{w}_k^t, \theta_k^t, k \in C_t)$
end for
$\mathcal{D}^s = \{\mathcal{D}_k^s\}_{k=1}^K \leftarrow$Collecting $\mathbf{x}_p$ generated by k-th client use Eq (9), where $\mathbf{x}_p = \mathbf{x}_s + \mathbf{n}$

Local_FeatDis($\mathbf{w}^t, \theta^t, \sigma_s^2$):
for each local epoch e with $e = 1, \cdots, E_d$ **do**
  $\mathbf{w}_k^{t+1}, \theta_k^{t+1} \leftarrow$ SGD update use Eq (9).
end for
Return $\mathbf{w}_k^{t+1}, \theta_k^{t+1}$ to server

Figure 3: FedFed algorithm for feature distillation [3].

Algorithm 2 FedAvg/FedProx with FedFed

Server Input: initial global model ϕ^0, communication round T_r.
Client k's Input: local epochs E_r, local private datasets $\mathcal{D}^k$, learning rate η_k.
Initialization: server distributes the initial model ϕ^0 to all clients,
Generate globally shared dataset $\mathcal{D}^s$. $\leftarrow$ Detail in Algorithm 1
Distribute $\mathcal{D}^s$ to all clients and $\mathcal{D}_t^k = \mathcal{D}^k \cup \mathcal{D}^s$.
Server Executes:
for each round $r = 1, 2, \cdots, T_r$ **do**
  server samples a subset of clients $C_r \subseteq \{1, ..., K\}$
  server **communicates** ϕ^r to selected clients $k \in C_r$
  for each client $k \in C_r$ **in parallel do**
    $\phi_k^{r+1} \leftarrow$ Client_Training(k, ϕ^r)
  end for
  $\phi^{r+1} \leftarrow AGG(\phi_k^{r+1})$
end for

Client_Training(k, ϕ^r):
ϕ^r initialize local model ϕ_k^r
for each local epoch e with $e = 1, 2, \cdots, E_r$ **do**
  $\phi_k^{r+1} \leftarrow$ SGD update with $\mathcal{D}_t^k$ using Eq. (10), FedProx uses Eq. (11)
end for
Return ϕ_k^{r+1} to server

Figure 4: Algorithm for FedFed [3].

In certain situations, FedFed can significantly increase the accuracy – one over 40% – and convergence rate of the model. FedFed didn't improve the model when FL already performed similarly to centralized learning or the noise necessary to preserve privacy was very large.

2.3 Poisoning Attack Protection [4]

Although FL is advertised as a secure method for sharing privacy-protected data, the presence of sensitive data as well as frequent communications with many decentralized parties leaves them vulnerable to attacks. Malicious entities may attempt to submit poisonous gradients which affect the model in one of two ways. They may either be targeted attacks, which attack particular labels, or untargeted attacks, which seek to lower the overall performance of the model. The attacks are difficult to detect because the gradients are encrypted and are not i.i.d. There are many ways to increase privacy, but they tend to struggle in this setting and bog down the model. Yazdinejad et al. seek to build a privacy protection scheme which protects against poisoning attacks without sacrificing the computation time, communication time, or accuracy of the model.

For malicious gradient detection, Yazdinejad et al. use both Gaussian Mixture Models (GMMs) and Mahalanobis Distance. The goal is to detect unusual gradients which are likely to be malicious. GMMs by design are good at identifying clusters of (normally distributed) data in heterogeneous datasets. Mahalanobis Distance measures the distance between a point and a distribution. It is scale-invariant and accounts for covariance within data. Before trying to identify malicious gradients, the data is split into a train and test set. To train the GMM model, first the parameters are trained using Expectation-Maximization (EM). Then the optimal number of clusters is chosen to minimize Baysian information criteria (BIC). Next, the Mahalanobis Distance is calculated for each group k (benign or malicious) for each gradient x_i using $MD(x_i) = \sqrt{(x_i - \mu_k)^\top \Sigma_k^{-1} (x_i - \mu_k)}$ where μ is the mean vector and Σ_k is the covariance matrix. The distribution of the Mahalanobis Distances is used to set a threshold. Finally, the gradient detection performance is tested on the test set, and if the performance is not satisfactory then the model is fine-tuned.

Yazdinejad et al. build redundancy into the auditing process to improve security. Auditors are the main trusted security entity which hold the encryption keys. All gradients are passed through the auditor before being aggregated on the central server. In this case, each gradient is inspected by multiple auditing entities. Redundancy added in one of two ways. In independent auditing, each auditing entity acts independently to validate gradients. In consensus auditing, the auditing entities must come to a consensus about the gradient in order to validate it.

Multiple techniques are also applied which work to both protect the model and improve performance. Stochastic gradient descent with momentum is applied during local training, which takes previous gradients' trajectories into account. This not only improves convergence rate and reduces variance, but also reduces the influence of unusual gradients. They also remove redundant gradients before aggregating. This reduces the number of communications. It also allows the model to schedule communications from clients to give time for more training steps between communications. Finally, they create an encryption algorithm with computes the least-computationally expensive encryption necessary to maintain privacy.

The proposed model greatly outperforms competitors at identifying targeted and untargeted attacks with a pessimistic 50% attack rate.

3 Simulation

3.1 Simulation Objective

As a rapidly evolving field, Federated Learning (FL) has several foundational frameworks. In this project, we focus on simulating two of them: FedAvg [1] and FedProx [8]. We aim to evaluate the efficiency of the FL framework, the sensitivity of these two FL frameworks to hyperparameters, and the robustness of the frameworks when having different kinds of heterogeneities.

To be more specific, we aim to investigate the following specific questions through simulation:

1. Basically, can FedAvg and FedProx address the issue of data privacy without compromising model accuracy?
2. How sensitive are FL methods to both framework-level and model-specific hyperparameters? Recall the algorithm of FedAvg (Figure 1) and FedProx (Figure 5), the hyperparameters include the fraction of participating clients per round, the penalty coefficient in FedProx, the number of local training epochs, and the number of global training rounds.

Algorithm 2 `FedProx` (Proposed Framework)

Input: $K, T, \mu, \gamma, w^0, N, p_k, k = 1, \cdots, N$
for $t = 0, \cdots, T-1$ **do**
  Server selects a subset S_t of K devices at random (each device k is chosen with probability p_k)
  Server sends w^t to all chosen devices
  Each chosen device $k \in S_t$ finds a w_k^{t+1} which is a γ_k^t-inexact minimizer of: $w_k^{t+1} \approx \arg\min_w \; h_k(w;\, w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$
  Each device $k \in S_t$ sends w_k^{t+1} back to the server
  Server aggregates the w's as $w^{t+1} = \frac{1}{K}\sum_{k \in S_t} w_k^{t+1}$
end for

Figure 5: Algorithm of FedProx

3. How robust are these two FL frameworks in the presence of data heterogeneity?

   Based on a review of several recent influential papers [8, 1], heterogeneity in federated learning (FL) can generally be categorized into two major types. The first is system heterogeneity, which refers to differences in storage, computational power, and communication capabilities across local devices in the federated network. Such differences often arise due to variability in hardware (e.g., CPU, memory), network conditions (e.g., 3G, 4G, 5G, Wi-Fi), and power constraints (e.g., battery level). These system-level variations significantly intensify challenges such as straggler mitigation and fault tolerance. The second type is statistical heterogeneity, which refers to the data not being identically distributed across local clients.

   In this report, I will conduct some simple simulations to reflect both types of heterogeneity. Details can be found in the Simulation Design part below.

3.2 Model and Dataset

As FL is a broad and flexible framework that can be integrated with various algorithms and methodologies across domains to achieve data privacy, I will first introduce the model and datasets used in my simulations before presenting the simulation design. I choose the Convolutional Neural Network (CNN) [9] as the underlying model, along with two widely studied datasets in the field of computer vision — MNIST [9] and CIFAR-100 [10].

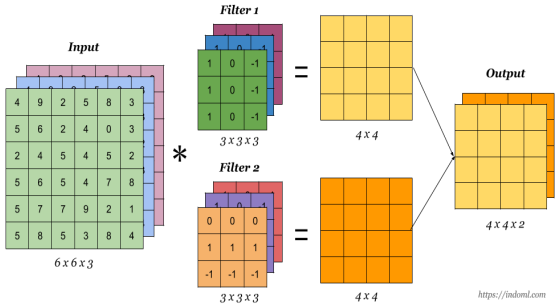


Figure 6: CNN Architecture

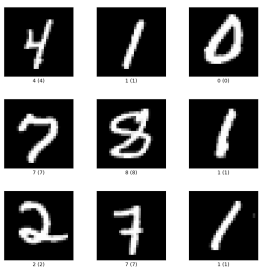


Figure 7: MNIST

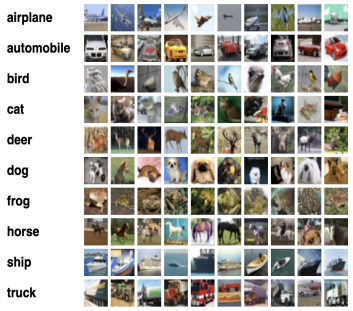


Figure 8: CIFAR100

CNN is the backbone of modern computer vision (CV), which can automatically extract spatial features from images through convolutional layers. Figure 6 shows the basic architecture of CNN. MNIST is the classic benchmark dataset in CV, including 70000 handwritten digit images, and is widely used for evaluating image classification models. CIFAR-100 is another widely used benchmark dataset in computer vision, consisting of 60000 color images spanning 100 object categories. Compared to MNIST, it presents a more challenging setting for evaluating image classification models due to its greater label granularity and image complexity. Figure 6 and 8 present an overview of the visual characteristics of our datasets.

3.3 Simulation Design

To address Objective 1 of this study, I use the centralized CNN model as a benchmark during the sensitivity analysis. Therefore, no separate experiment is designed specifically for this objective. The experimental designs for the other two objectives are detailed below.

It is also worth noting that, to fully investigate all three objectives of this study, particularly the sensitivity analysis to various parameters, a comprehensive set of simulation configurations would ideally be required. However, due to the substantial computational cost of each FL simulation, this report adopts a more simplified and localized experimental design. Despite this limitation, the results still offer meaningful insights and are valuable for comparative analysis.

Additionally, as a deep learning model, CNN is highly sensitive to hyperparameters such as learning rate and network architecture. Extensive prior work has examined CNN tuning on both MNIST and CIFAR datasets [11, 12, 13]. Since the primary focus of this study lies in the FL framework itself, we intentionally simplified the hyperparameter tuning process. Accordingly, the analysis emphasizes comparative trends over absolute performance metrics.

3.3.1 Sensitivity

Due to the long runtime required for a single FL simulation — often lasting several hours — I primarily use the MNIST dataset and simulate under the IID local data setting for addressing Objectives 2.

Recalling the original papers that proposed the FedAvg and FedProx frameworks, we can identify the key hyperparameters associated with each. In FedAvg, the most critical hyperparameter is the fraction of local clients participating in each round of training. In FedProx, an additional important hyperparameter is the coefficient μ of the proximal term in the loss function.

- To study the sensitivity of the fraction of participating clients in the FedAvg framework, I simulate the IID setting using FedAvg while keeping all other parameters fixed. I vary the client participation rate from 0.05 to 0.7 and record the test accuracy and test loss after 10 rounds of global updates.
- For the μ parameter in FedProx, I similarly simulate under the IID setting using the FedProx framework. With all other parameters fixed, I vary μ from 0.001 to 1.0 and record the test accuracy and test loss after 20 rounds of global updates.
- I also conduct an evaluation using the MNIST dataset to assess the impact of the number of global and local epochs on model performance in FL. Specifically, I compare settings where the number of global and local epochs are set to either 10 or 20, and examine model performance in the testing dataset.

3.3.2 Robustness

To evaluate the robustness of the two FL frameworks, I primarily follow the simulation approach outlined in Paper [8]. I simulate system heterogeneity by introducing a parameter called the "straggler rate", which represents the proportion of local clients that are unable to complete their full local training due to hardware limitations, communication costs, or other system-level issues. These clients return only partially trained updates during each communication round.

For clients with only partial local results, the two frameworks adopt different strategies. FedAvg simply drops these clients from the global aggregation in that round, while FedProx incorporates their updates entirely-assuming that the local training was performed using the loss function augmented with the proximal term.

Statistical heterogeneity is introduced by controlling the way data is allocated to local clients. Instead of assigning training data randomly, I implement a label-based partitioning strategy. Specifically, using the MNIST dataset as an example, I first sort the entire dataset by label, then divide it into 200 equal parts. Since MNIST contains 60,000 samples with approximately uniformly distributed labels, each part contains around 300 training samples of nearly the same label. Then, for a simulation with K local clients, I assign $\frac{300}{K}$ of these parts to each client. For simplicity, here I assume all clients receive equal amounts of data.

As previously mentioned, there are numerous hyperparameters to consider when using FL frameworks. Although the primary goal at this stage is not to achieve the highest accuracy, but rather to compare the performance and suitability of the two frameworks and to examine their robustness under heterogeneity, I still performed some basic hyperparameter tuning.

- First, I conducted learning rate tuning on both the MNIST and CIFAR-100 datasets, as it is a key hyper-parameter.

  For MNIST, I fixed the number of local epochs at 3 and global epochs at 12. Based on existing literature, the batch size was set to 10, and the learning rate was selected from the set 0.01, 0.03, 0.001 based on performance on the test set.

  For CIFAR-100, following similar literature guidance, both the learning rate and batch size were tuned, with candidates chosen from 0.01, 0.03, 0.001 and 16, 32, respectively. In this process, I set the number of local epochs to 2 and global epochs to 15.

  In summary, I selected lr = 0.01 and batch size = 10 for MNIST, and lr = 0.01 and batch size = 16 for CIFAR-100.

- A similar tuning strategy was applied for the μ parameter in FedProx. In this case, both system and statistical heterogeneity were assumed—specifically, local data was set to be non-IID, and 50% of local clients were configured to return partial updates in each round.

  For MNIST, the number of local epochs was fixed at 3, global epochs at 12, and the learning rate was set to the tuned value mentioned above.

  For CIFAR-100, consistent with the previous setup, the number of local epochs was set to 2 and global epochs to 15, with the learning rate and batch size set to the selected optimal values.

  The μ parameter was tuned by selecting from a limited candidate set $\{0.001, 0.01, 0.1, 1\}$. As a result, the optimal μ was found to be 0.1 for MNIST and 0.01 for CIFAR-100.

All other parameters and model structure are set based on findings from previous literature [1, 8, 14] and the sensitivity analysis results. For both IID and non-IID settings (indicating the absence and presence of statistical heterogeneity), and under different straggler rates (0.0 and 0.5, representing the absence and presence of system heterogeneity), I ran 50 and 100 rounds of training and global updates for the MNIST and CIFAR-100 datasets, respectively, and recorded the results. All parameter settings used in this simulation stage are summarized in Table 1.

Table 1: Parameter Settings for Robustness Simulation

Dataset	Local Distribution	straggler rate	learning rate	batch size	$\mu_{[FedProx]}$	global epoch
MNIST	{IID, Non-IID}	$\{0.0, 0.5\}$	0.01	10	0.1	50
CIFAR100			0.01	16	0.01	100

Note: local epoch = 10, fraction rate = 0.1 for all of these settings.

3.4 Evaluation Metric

In this report, we report all metrics based on the global objective $f(w)$. The primary evaluation metrics are the test loss and test accuracy, which are computed as follows: after each round of local training and global aggregation, the updated global model is evaluated on the test dataset across all local clients, and the results are then averaged to obtain the final metrics.

In addition, to provide confidence intervals for the accuracy metric and enable more rigorous statistical inference, I apply Monte Carlo (MC) Dropout during evaluation. Specifically, for each test-time evaluation, I perform 50 stochastic forward passes using MC Dropout, resulting in 50 accuracy estimates per evaluation round. These repeated measurements allow us to compute confidence intervals for the accuracy. Due to the relatively high computational cost of each forward pass, the number of repetitions is limited to 50 for practical reasons.

Also, following the approach in [8], I use the trend of training loss and training accuracy in terms of rounds as secondary evaluation metrics.

3.5 Simulation Result

Prior to the presentation and discussion of results, it should be clarified that this report does not aim to compare model performance on the two datasets against state-of-the-art baselines. The simulations are intended

to assess the effectiveness of the FL framework and to compare the sensitivity and robustness of two representative FL algorithms.

3.5.1 Sensitivity Analysis

Table 2 shows the results for the sensitivity analysis of FedAvg in terms of the parameter "The Fraction of Local Clients Each Round". According to the results, we can see that when there is no heterogeneity, which means the local data is IID and all local clients are good at computing, the FedAvg CNN has great performance, with a similar test accuracy (97%) compared with centralized CNN (98%), which means the FedAvg framework can help address the data privacy issues without any loss of model performance. Also, under this ideal situation, FedAvg will keep the stability when choosing different fractions of local clients to participate in computing in each round.

Table 2: Sensitivity Analysis of FedAvg CNN in terms of Fraction Rate

Dataset	Model	Parameter	Value	Test Loss	Test Accuracy
MNIST	FedAvg CNN	fraction rate	0.05	7.12	0.97
			0.1	7.39	0.97
			0.3	7.54	0.97
			0.5	6.97	0.97
			0.7	6.98	0.97
MNIST	Benchmark: Centralized CNN with same parameters			3.80	0.98

Note: Other hyperparameters used: lr = 0.01, local epoch = 10, global epoch = 10, IID = 1.

Table 3 shows the results of sensitivity analysis in terms of μ in FedProx, where the values in square brackets mean the 95% confidence intervals. Based on the results, we observe that FedProx is highly sensitive to the choice of the μ parameter. Given that one of the primary goals of FedProx is to address system heterogeneity, I conducted simulations under straggler rates of 0.0, 0.5, and 0.9. The findings show that regardless of the degree of system heterogeneity, the performance of FedProx is significantly influenced by the μ value.

In addition, the analysis of FedProx also reveals the following insights:

(1) In ideal scenarios without any heterogeneity (neither system nor statistical), FedProx underperforms compared to FedAvg.

(2) When only system heterogeneity is present, FedProx can outperform FedAvg with an appropriately tuned μ, particularly under higher levels of system heterogeneity.

A more detailed comparison of the performance and robustness of FedAvg and FedProx under heterogeneous settings is provided in the following subsection.

Table 3: Sensitivity Analysis of FedProx CNN: in terms of μ

Dataset	Straggler rate	Model	Parameter	Value	Test Accuracy	Test Loss
MNIST	0.0	FedProx CNN	μ	0.0	0.963 [0.9626, 0.9633]	0.148 [0.1466, 0.1496]
				0.001	0.965 [0.9647, 0.9655]	0.131 [0.1301, 0.1328]
				0.01	0.964 [0.9637, 0.9643]	0.133 [0.1319, 0.1341]

Dataset	Straggler rate	Model	Parameter	Value	Test Accuracy	Test Loss
				1.0	0.851 [0.8508, 0.8521]	0.551 [0.5504, 0.5520]
		Benchmark: FedAvg CNN			0.966 [0.9659, 0.9664]	0.126 [0.1253, 0.1275]
MNIST	0.5	FedProx CNN	μ	0.0	0.963 [0.9631, 0.9638]	0.134 [0.1329, 0.1354]
				0.001	0.964 [0.9635, 0.9642]	0.135 [0.1338, 0.1361]
				0.01	0.967 [0.9663, 0.9671]	0.129 [0.1277, 0.1302]
				1.0	0.859 [0.8582, 0.8594]	0.584 [0.5825, 0.5845]
		Benchmark: FedAvg CNN			0.956 [0.9557, 0.9565]	0.171 [0.1691, 0.1722]
MNIST	0.9	FedProx CNN	μ	0.0	0.962 [0.9621, 0.9628]	0.137 [0.1357, 0.1379]
				0.001	0.963 [0.9624, 0.9631]	0.136 [0.1351, 0.1372]
				0.01	0.962 [0.9619, 0.9626]	0.136 [0.1349, 0.1373]
				1.0	0.889 [0.8881, 0.8891]	0.429 [0.4277, 0.4294]
		Benchmark: FedAvg CNN			0.886 [0.8856, 0.8870]	0.527 [0.5230, 0.5309]

Note: Other hyperparameters are set as: learning rate = 0.01, local epoch = 10, global epoch = 10, frac rate = 0.1, IID = 1.

Table 4 presents the sensitivity analysis results with respect to Local Epoch for both FedAvg and FedProx. The values in square brackets also represent the 95% confidence intervals. Based on the findings, both frameworks appear relatively insensitive to changes in the number of local epochs on the locally IID MNIST dataset. During our literature review, we found that the original FedAvg paper [1] also discussed this issue, suggesting that excessively large local epochs may lead local updates to deviate from the optimal optimization direction, particularly under non-IID conditions [8]. However, within a reasonable range, the number of local epochs does not significantly affect model performance.

Our results further indicate that Global Epoch also has limited influence in this setting. Considering the simplicity of the MNIST dataset, the model already achieves stable performance when the number of global epochs is set to 10. Increasing it to 20 does not lead to noticeable improvements. Nonetheless, for more complex datasets, carefully selecting an appropriate number of global epochs remains important.

Table 4: Sensitivity Analysis of Local Epoch

Dataset	Model	Global Epoch	Local Epoch	Test Accuracy	Test Loss
MNIST	FedAvg CNN	10	10	0.966 [0.9659, 0.9664]	0.126 [0.1253, 0.1275]
		10	20	0.964 [0.9638, 0.9645]	0.156 [0.1540, 0.1575]
		20	10	0.972 [0.9713, 0.9720]	0.112 [0.1104, 0.1130]
		20	20	0.968 [0.9673, 0.9679]	0.163 [0.1609, 0.1647]
	FedProx CNN	10	10	0.967 [0.9663, 0.9671]	0.129 [0.1277, 0.1302]
		10	20	0.967 [0.9666, 0.9672]	0.132 [0.1309, 0.1334]
		20	10	0.972 [0.9722, 0.9728]	0.108 [0.1067, 0.1093]
		20	20	0.969 [0.9691, 0.9698]	0.134 [0.1322, 0.1351]

Note: FedAvg hyperparameters: learning rate = 0.01, local epoch = 10, global epoch = 10, frac rate = 0.1, IID = 1.
Note: FedProx uses the same settings, with additional straggler rate = 0.5 and $\mu = 0.01$.

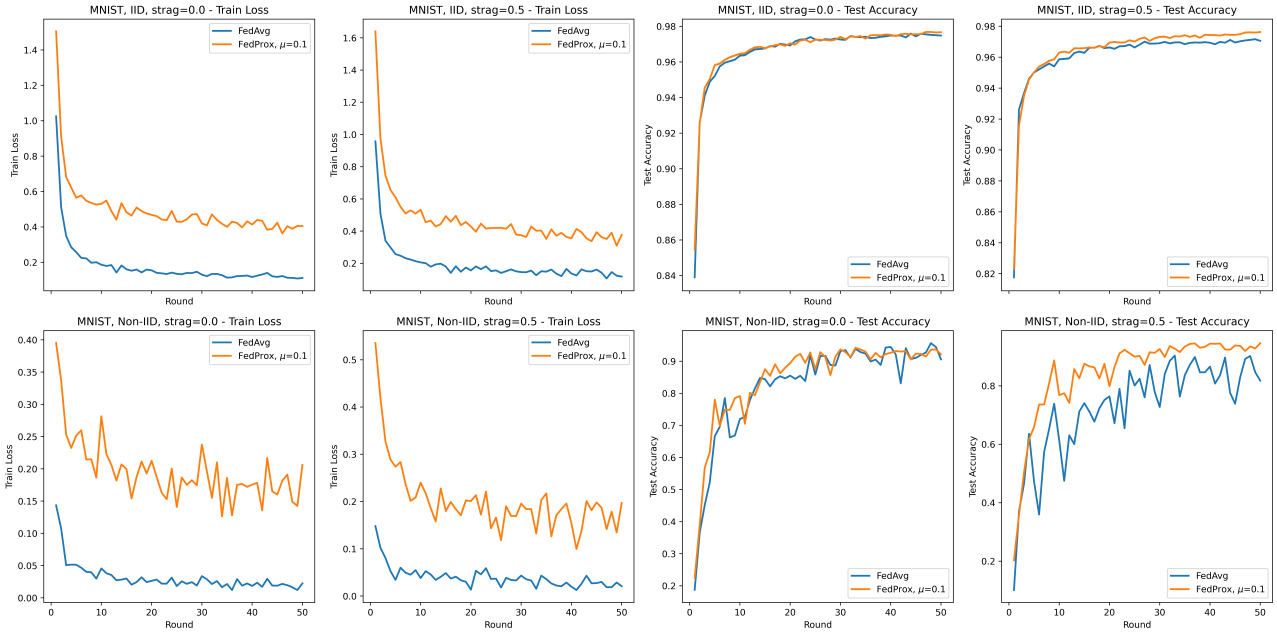
In summary, our simulation results indicate that the FedAvg framework is relatively insensitive to both the fraction of local clients participating in each round and the number of local epochs, while it is more sensitive to the penalty parameter μ in its loss function.

3.5.2 Robustness Analysis

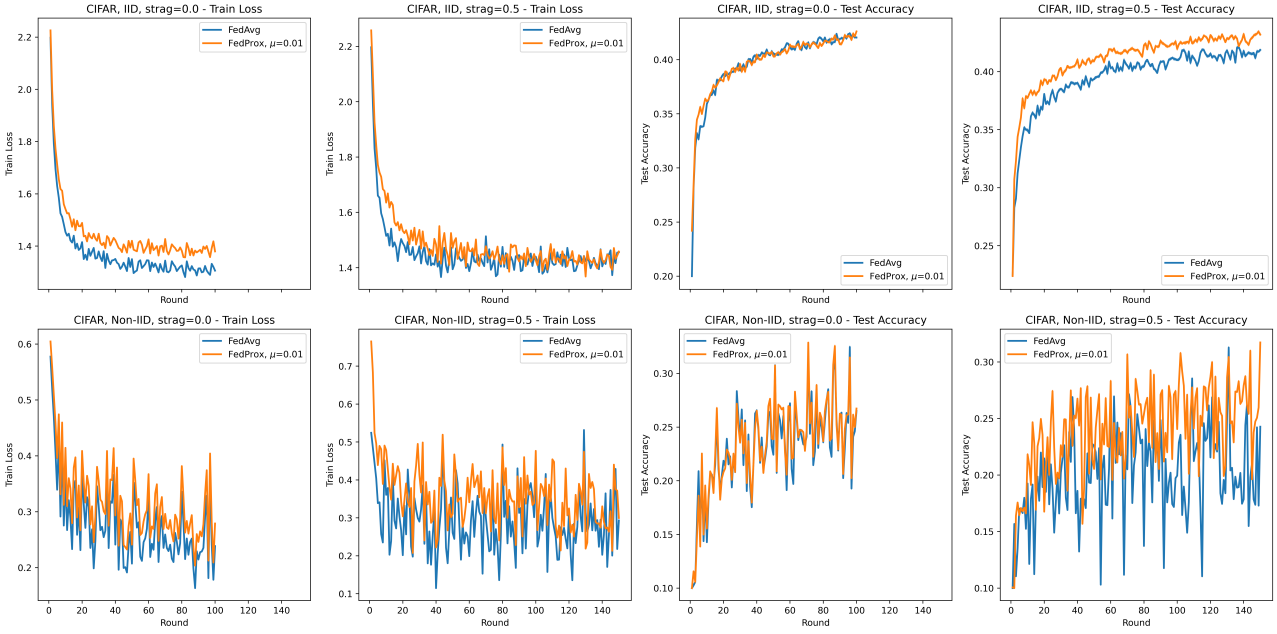
As previously discussed, the robustness analysis section of this report aims to compare the robustness of FedAvg and FedProx under conditions of both system and statistical heterogeneity. In addition to the relatively simple MNIST dataset, we also conduct a more challenging evaluation using CIFAR-100, with results presented in Figures 9a and 9b. The test accuracy curves in the figures include confidence intervals; however, due to the relatively low dropout rate (set to 0.3) and the inherently optimization-driven nature of deep learning, the confidence intervals are not visually prominent on the curves. Therefore, Table 5 provides the final test accuracy and loss values along with their corresponding standard deviations to more clearly reflect performance variability.

According to the result, we can get some insights:

- In the absence of any heterogeneity, FedProx demonstrates weaker training performance compared to FedAvg, while both frameworks achieve similar predictive performance on the test set.
- When only system heterogeneity exists—i.e., IID local data with a nonzero straggler rate (0.5 in our simulation)—FedProx still shows weaker training performance, but significantly outperforms FedAvg on the test set, suggesting that the proximal term effectively mitigates overfitting.
- When only statistical heterogeneity exists—i.e., Non-IID local data but straggler rate = 0.0 (all clients perform computation and communication reliably)—both methods achieve comparable test accuracy, but FedProx produces more stable predictions with lower variance, particularly on the CIFAR-100 dataset.



(a) Robustness Analysis on MNIST



(b) Robustness Analysis on CIFAR-100

Figure 9: Robustness analysis Result

- When both system and statistical heterogeneity are present, FedProx achieves notably higher prediction accuracy and improved stability compared to FedAvg.

Overall, our simulation results suggest that FedProx is more robust to heterogeneity, especially in scenarios where both system and statistical heterogeneity are present. While predictive performance degrades under such challenging conditions, FedProx consistently yields more stable predictions with reduced variance, compared with FedAvg.

Table 5: Test Accuracy and Loss for Robustness Analysis

Dataset	IID/ Non-IID	Straggler Rate	Model (with μ)	Global Epoch	Test Accuracy (with std)	Test Loss (with std)
MNIST	IID	0.0	FedAvg	50	0.974 (0.0001)	0.116 (0.0008)
		0.0	FedProx (μ=0.1)		0.977 (0.0002)	0.099 (0.0007)
		0.5	FedAvg		0.971 (0.0002)	0.125 (0.0008)
		0.5	FedProx (μ=0.1)		0.976 (0.0001)	0.093 (0.0006)
	Non-IID	0.0	FedAvg		0.906 (0.0002)	0.339 (0.0007)
		0.0	FedProx (μ=0.1)		0.922 (0.0003)	0.231 (0.0006)
		0.5	FedAvg		0.818 (0.0003)	0.669 (0.0013)
		0.5	FedProx (μ=0.1)		0.946 (0.0002)	0.175 (0.0005)
CIFAR	IID	0.0	FedAvg	100	0.420 (0.0006)	1.616 (0.0010)
		0.0	FedProx (μ=0.01)		0.427 (0.0005)	1.634 (0.0009)
		0.5	FedAvg	150	0.419 (0.0004)	1.644 (0.0010)
		0.5	FedProx (μ=0.01)		0.432 (0.0005)	1.592 (0.0008)
	Non-IID	0.0	FedAvg	100	0.265 (0.0004)	2.159 (0.0007)
		0.0	FedProx (μ=0.01)		0.268 (0.0003)	2.135 (0.0007)
		0.5	FedAvg	150	0.242 (0.0003)	2.391 (0.0006)
		0.5	FedProx (μ=0.01)		0.317 (0.0004)	1.813 (0.0007)

4 Discussion

In this report, we introduced the motivation, foundational framework, and design principles of Federated Learning (FL), along with its applications in areas such as large language models (LLMs) and healthcare. In the literature review, we also examined recent advancements in the field, including Low-Parameter Federated Learning and the FedFed algorithm. Finally, we conducted numerical simulations of two widely used FL frameworks—FedAvg and FedProx—using CNN models on MNIST and CIFAR-100. Our findings highlight why FL has gained increasing attention across various domains and demonstrate its strong potential for real-world deployment. Simulation results show that in certain scenarios, FL can effectively preserve data privacy without compromising model performance. In more heterogeneous settings, while FL may not achieve the same accuracy as centralized approaches, ongoing research efforts, such as the transition from FedAvg to FedProx, continue to enhance the robustness and reliability of FL frameworks.

We also explored the current open problems and emerging trends in the field of Federated Learning. A survey paper by Wen et al. (2023) gives several avenues for improvement in FL [15]. For instance, new applications for FL should be explored, such as education, e-government, meteorology, coal, and power dispatching. They also suggest improvements for security, noting that the fight between privacy and attacks is a technological arms race, and that security measures can limit usage of the data. They also give several more specific areas for improvement. First, that client selection should be improved. The algorithm must select high-quality, stable clients out of very large number of possibilities before each iteration. Finally, they suggest fusion models for fusing multiple types of data, such as text and images.

For FedFed, Yang et al. [3] suggest exploring applications in recommendation systems and healthcare systems. They acknowledge that FedFed can be computationally expensive and is vulnerable to attacks, so more work is needed to improve computational overhead and security. Yazdinejad et al. [4] suggest exploring applications in smart city infrastructure and Internet of Things networks. They plan to extend the model's privacy protection to quantum computing attacks and collaborative attacks to keep pace with the growing complexity of attacks.

References

[1] 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.

[2] Jingang Jiang, Haiqi Jiang, Yuhan Ma, Xiangyang Liu, and Chenyou Fan. Low-parameter federated learning with large language models. In *International Conference on Web Information Systems and Applications*, pages 319–330. Springer, 2024.

[3] Zhiqin Yang, Yonggang Zhang, Yu Zheng, Xinmei Tian, Hao Peng, Tongliang Liu, and Bo Han. Fedfed: Feature distillation against data heterogeneity in federated learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 60397–60428. Curran Associates, Inc., 2023.

[4] Abbas Yazdinejad, Ali Dehghantanha, Hadis Karimipour, Gautam Srivastava, and Reza M. Parizi. A robust privacy-preserving federated learning model against model poisoning attacks. *IEEE Transactions on Information Forensics and Security*, 19:6693–6708, 2024.

[5] Heiko Ludwig and Nathalie Baracaldo. *Federated learning: A comprehensive overview of methods and applications*. Springer, 2022.

[6] Hao Guan, Pew-Thian Yap, Andrea Bozoki, and Mingxia Liu. Federated learning for medical image analysis: A survey. *Pattern Recognition*, page 110424, 2024.

[7] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[8] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

[11] Mikolaj Wojciuk, Zaneta Swiderska-Chadaj, Krzysztof Siwek, and Arkadiusz Gertych. Improving classification accuracy of fine-tuned cnn models: Impact of hyperparameter optimization. *Heliyon*, 10(5), 2024.

[12] Eva Tuba, Nebojša Bačanin, Ivana Strumberger, and Milan Tuba. Convolutional neural networks hyperparameters tuning. In *Artificial intelligence: theory and applications*, pages 65–84. Springer, 2021.

[13] Ramaprasad Poojary, Roma Raina, and Amit Kumar Mondal. Effect of data-augmentation on fine-tuned cnn model performance. *algorithms*, 5:6, 2021.

[14] Claire Hyeju Lee. Investigating cnns performance on the cifar-10 dataset through hyperparameter tuning. 2025.

[15] Jie Wen, Zhixia Zhang, Yang Lan, Zhihua Cui, Jianghui Cai, and Wensheng Zhang. A survey on federated learning: challenges and applications. *International Journal of Machine Learning and Cybernetics*, 14(2):513–535, 2023.