Heterogeneous Federated Learning

Student's Name: Ayana & Bharti Bansal
Roll Number: 2022123 & 2022136

BTP report submitted in partial fulfillment of the requirements
for the Degree of B.Tech. in Electronics & Communication Engineering
on 3rd May, 2025

BTP Track: Research

BTP Advisor
Dr. Ranjitha Prasad

Indraprastha Institute of Information Technology
New Delhi

Student's Declaration

We hereby declare that the work presented in the report entitled **"Heterogeneous Federated Learning"** submitted by us for the partial fulfillment of the requirements for the degree of *Bachelor of Technology* in *Electronics & Communication Engineering* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of our work carried out under the guidance of **Dr. Ranjitha Prasad**. Due acknowledgements have been given in the report to all material used. This work has not been submitted anywhere else for the reward of any other degree.

Student's Names: Ayana(2022123) & Bharti Bansal(2022136)
Place & Date: New Delhi, 3rd May, 2025

Certificate

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Dr. Ranjitha Prasad **Place & Date: New Delhi, 3rd May, 2025**

Abstract

This research extends our previous work on federated learning (FL) by incorporating the FedProx algorithm, with a focus on addressing system and statistical heterogeneity in collaborative training scenarios. Building on earlier experiments with FedAvg, we now explore how proximal regularization and partial local updates improve model convergence and stability in heterogeneous environments. Using the MNIST dataset and two clients with non-IID data and varying computational capabilities, we evaluate FedProx's ability to maintain accuracy under non-uniform local work. Our study highlights the theoretical and algorithmic advancements introduced by FedProx and provides empirical evidence of its robustness and efficiency compared to FedAvg. These findings offer deeper insights into the trade-offs between convergence, accuracy, and resource variability in real-world FL deployments.

Keywords: Federated Learning (FL), Federated Averaging (FedAvg), Software Defined Radio(SDR), Universal Software Radio Peripheral(USRP), Independent and Identically Distributed (IID), Non-Independent and Identically Distributed (Non-IID), Epochs, MultiLayer Perceptron(MLP)

Acknowledgments

We would like to begin by expressing our heartfelt gratitude to our project advisor, Dr. Ranjitha Prasad, for her unwavering support, insightful guidance, and encouragement throughout this project. Her expertise and continuous motivation have been essential to the completion of this work, and we are thankful for the opportunity to work under her mentorship.

Our sincere thanks go to PhD student Zubair Shaban, whose valuable support and guidance have helped us overcome numerous challenges. We are also grateful to Varun Muttepawar and Arjun Gupta, whose collaborative spirit and ideas greatly enriched the progress of this project.

Lastly, we would like to express our deepest appreciation to our family for their constant love, encouragement, and belief in us. Their support has been a pillar of strength throughout this journey. We extend our gratitude to everyone who has been part of this experience and contributed to its success.

Work Distribution

We equally contributed to the BTP Heterogeneous Federated Learning project, collaborating on all aspects of the literature review, code, and analysis.

Contents

Chapter 1

Introduction to Federated Learning

1.1 Understanding of FedAvg

FedAvg is a widely adopted algorithm in FL that coordinates the distributed training of machine learning models across multiple clients while ensuring data privacy. It combines locally computed model updates from client devices to form a global model without transferring raw data to a central server.

- **Decentralized Data Training**: Each client trains the model locally on its private data. This ensures that sensitive data remains on the client device.
- **Model Aggregation**: Instead of sharing data, clients send model updates (e.g., gradients or weights) to a central server, which aggregates them to update the global model.
- **Communication Efficiency**: By iterating several times locally before sharing updates with the server, FedAvg reduces communication overhead compared to traditional distributed training methods.
- **Privacy Preservation**: Since raw data never leaves the client devices, privacy risks associated with data transfer are mitigated.

1.2 Algorithm of FedAvg

FedAvg algorithm is a cornerstone method in federated learning, designed to train a global machine learning model across decentralized data without sharing raw data among clients.

Here are the steps for doing FedAvg:

1. **Initialization and Client Selection:** The central server initializes a global model and selects a subset of clients to participate in the training round.

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**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $m_t \leftarrow \sum_{k \in S_t} n_k$
  $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$ *// Erratum*[4]

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

2. **Model Distribution and Local Updates:** The server sends the global model to the selected clients. Each client trains the model locally on its private dataset for E epochs using its local loss function and computes updated model parameters.

3. **Upload and Aggregation:** The clients send their updated models back to the server. The server aggregates the updates using a weighted averaging scheme to produce the new global model.

4. **Repeat Until Convergence:** The server iteratively repeats the steps of client selection, model distribution, local updates, and aggregation for a fixed number of rounds or until the model achieves satisfactory performance.

1.3 Dataset Description

To demonstrate how the parameters of various models running on different clients can be combined using the FedAvg method, we implemented a straightforward experiment using the MNIST dataset. The MNIST dataset consists of images of handwritten digits and is commonly used for testing machine learning models. We conducted testing on both IID and non-IID data, allowing us to observe the impact of data distribution on model performance. For the IID setup, the dataset was divided evenly across clients, ensuring each client had an equal representation of all digits. In contrast, the non-IID setup introduced imbalanced distributions, with some clients receiving a disproportionate number of certain digits. We used a Multi-Layer Perceptron model for classification, and through this testing, we analyzed accuracy trends, observing how the model's performance varied based on the data distribution across clients.

Chapter 2

SDR and ZigBee Based Wireless TestBed

2.1 Introduction of SDR and ZigBee

Our hardware setup consists of four main components: the central server, Client 1, Client 2, and the ZigBee. Each component is equipped with the necessary hardware for communication. The Xbee S2C devices are used for network-level synchronization, while the physical layer communication is handled by the NI USRP 2900, using an omnidirectional VERT 2450 antenna.

To ensure robustness, the USRPs employ differential modulation, eliminating the need for channel estimation. Key signal specifications include a 2.5 GHz operating frequency, Root Raised Cosine pulse shaping, a reception gain of 20 dB, a 50 kHz IQ rate, a 6.25 k symbol rate (with 8 samples per symbol), and a default transmission power of 20 dB. Software-Defined Radio (SDR) is a communication system that uses software to define and control the behavior of the radio, including frequency, modulation, and signal encoding. This flexibility allows SDRs to be easily reconfigured for different radio tasks.

USRP is the hardware that enables a computer to interact with radio frequency signals. USRPs are often used in combination with SDRs to create a versatile platform for radio communication research and development. The USRP hardware consists of a motherboard, which handles processing and communication, and one or more daughterboards that provide the Radio frequency(RF) front-end for transmitting and receiving signals. These components are connected via high-speed interfaces like PCI Express or Gigabit Ethernet.

To use a USRP, we connect it to a computer running SDR software, such as LabVIEW by National Instruments, through a USB or Ethernet cable. Once connected, the USRP becomes a device on the computer, and the SDR software controls its behavior, allowing for transmission and reception of radio signals.

In summary, SDR software controls the radio's behavior, while the USRP hardware interfaces with radio signals. Together, they offer a powerful and adaptable platform for working with RF

signals.

2.2 Synchronization Process for Wireless FL

For effective operation of the wireless FL system, synchronization is critical. We used ZigBee for transmission scheduling to achieve this. Below is an overview of the synchronization steps:

1. **Transmission from Client 1:** Client 1 begins by sending its locally trained model parameters to the server within a specified time interval.
2. **Zigbee Acknowledgment from the Server:** Upon receiving the data, the server immediately acknowledges the reception via the Zigbee network, confirming a successful transfer.
3. **Client 2 Signaling:** Next, the server instructs Client 2 to send its locally trained model parameters, with a response expected in the time interval.
4. **Server Acknowledgment of Client 2 Data:** Once the server receives Client 2's data, it sends an acknowledgment over the Zigbee network, confirming the second transmission phase has concluded.
5. **Data Collection and Global Model Update:** The server aggregates the local model parameters from both clients to update the global model, confirming the process via Zigbee.
6. **Signal Broadcasting to Activate Receptors:** To synchronize the updated data, the server broadcasts a signal to both clients, instructing them to activate their receptors.
7. **End of Communication Round:** The round concludes when both clients independently confirm receipt of the updated data via the Zigbee network. The server stops broadcasting after receiving these confirmations.

NI USRP 2900 Specifications:

1. Frequency Range: 70 MHz to 6 GHz
2. Frequency Step: 1 kHz
3. Maximum Output Power: 20 dBm
4. Gain Range: 89.75 dB
5. Gain Step: 0.25 dB
6. Frequency Accuracy: 2.5 ppm
7. Maximum Real-Time Bandwidth: 56 MHz

8. Maximum I/Q Rate (Streaming): 15 MS/s
9. Maximum I/Q Rate (Burst): 61.44 MS/s
10. DAC Resolution: 12 bits

The NI USRP 2900 offers a wide frequency range, making it suitable for diverse applications. It can adjust its operating frequency with fine accuracy and provides high output power, enabling long-distance transmission in challenging environments. The USRP 2900 also features a wide gain range with small adjustments, useful for amplifying weak signals. Its high frequency accuracy and broad real-time bandwidth enable processing of complex signals in demanding conditions. Additionally, the device's high I/Q rate supports high-frequency operations, and the 12-bit DAC ensures precise analog signal output.

Chapter 3

Integration of Machine Learning with Hardware

3.1 Hardware Setup Using USRP

In this work, we integrated federated learning with hardware devices to create a functional experimental setup. We used three USRPs (Universal Software Radio Peripherals), where two served as a client transmitter and the other acted as the centralized server. A host PC was employed to program the USRPs using LabVIEW and Python.

In our proposed model, each client device ran a machine learning algorithm locally and transmitted its computed model updates to the server. These updates were adjusted by subtracting the global model parameter Θ_t, which the server computed as the average of the received updates. This adjustment helped normalize the transmitted data, significantly reducing quantization errors caused by large data variations.

To help explain the setup visually, We included diagrams illustrating the data flow between the server and clients (Figure 3.1.1) and the hardware arrangement in the lab (Figure 3.1.2).

3.2 Transmitting Machine Learning Parameters

The setup illustrates three USRP devices. USRP1 is configured as a transmitter for one client, USRP2 was configured for 2nd client while USRP3 functions as a server to receive signals. These USRPs were programmed using LabVIEW software to cater to the specific requirements of our federated learning experiment.

The machine learning task involves a Multi-Layer Perceptron model trained for using the MNIST dataset. The dataset was divided into subsets, one for each client. Both clients shared a common model, and training was conducted in a federated manner using a learning rate of 0.00001 over 20 communication rounds. For each round, the local models were trained for either 2 or 4

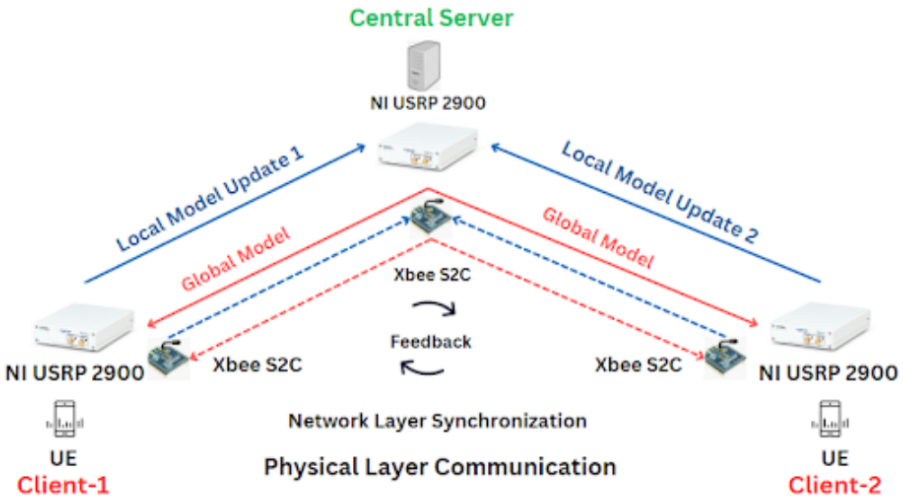

Figure 3.1.1: Proposed Block Diagram

Figure 3.1.2: Hardware Arrangement

epochs per client before aggregation. The loss function employed is the Mean Squared Error, optimized using MLP Classifier with 10 output classes. To ensure accurate transmission and synchronization between USRPs, sync bits (preamble) and guard bits were appended. These bits helped avoid overlap between data symbols and facilitated precise alignment. The QPSK-modulated symbols consisted of in-phase (I) and quadrature (Q) components, sampled at 200 kHz to ensure compatibility with LabVIEW's buffer storage limits.

The transmission process required specifying the IP address of the USRP and selecting the

appropriate channel (e.g., TX1 or RX2). The NI USRP-2900 used in our setup features two channels, both of which can act as receivers, while only one supports transmission. A carrier frequency of 2.5 GHz was used for our experiment.

The system is configured to transmit quantized weights and biases for one client at a time. To transmit data for the second client, the Python script path must be updated to generate a new set of model parameters. The NI USRP Packet Transmitter was employed to prepare multiple data packets containing the necessary synchronization, guard information, and the data.

3.3 Receiving Machine Learning Parameters

Server holds the global model, which is updated by averaging the weights and biases received from multiple clients. In our experiment, we produced results for two clients. The LabVIEW code used for receiving client data specifies the USRP's IP address, channel, center frequency, and IQ rate to match the transmitter's settings.

At the receiver end, sync bits are detected by identifying the peak in the auto-correlation function with the known reference data. Once synchronization is achieved, the data reception begins. The USRP Packet Receiver decodes the transmitted packets, demodulates the QPSK symbols back into bits, and passes these bits to a Python script. The script reconstructs the original weight and bias matrices.

After receiving matrices from both clients, the server computes the Federated Averaging by averaging these values to update the global model.

Chapter 4

System Heterogeneity

4.1 Understanding System Heterogenity

In the current FL study, there is a fundamental gap that has not been seen in traditional centralized ML paradigms, known as the system-heterogeneity issue. Specifically, we consider that the system-heterogeneous FL issue consists of two types of heterogeneity: data and device. The data-heterogeneity is also known as the non-i.i.d. training dataset. As the training samples on the remote devices(clients) are collected by the devices themselves based on their unique environment, the data distribution can vary heavily between different remote devices.
The device-heterogeneity stems from the heterogeneous FL network, where remote devices are in large numbers and have a variety of computational capacities. Specifically, for the partially participated FL scheme where each remote learning process is usually limited to a responding time, the diverged computational capacity can lead to heterogeneous local training updates e.g., the remote device with limited computational capacity is only able to return a non-finished update. In further sections, we analysed the effect of data heterogeneity(Non-IID dataset) and device heterogeneity(varying number of local training epochs) through simulations on global model accuracy and convergence.

4.2 IID vs Non-IID

4.2.1 IID Dataset

In our experiment, we used the MNIST dataset with an IID (Independent and Identically Distributed) data partition. In this setting, the dataset is split in such a way that each client receives a random and representative subset of the entire MNIST dataset, with each class being evenly distributed across all clients. This ensures that each client holds a balanced representation of the data, allowing for more efficient training since the data distribution across clients mirrors the overall dataset.

Using the MLP (Multi-Layer Perceptron) model, we trained on this IID partition of the MNIST dataset. The training process was conducted using the Federated Averaging (FedAvg) method, where model updates were averaged after each round of training across the selected clients. In the IID setup, the MLP model showed quick convergence, as the consistent and balanced distribution of the data helped each client contribute meaningful updates to the global model. This resulted in improved performance with faster convergence and higher accuracy compared to non-IID scenarios. Figure 4.2.1 Shows code for IID data Split.

```
# Load the entire dataset
dataset_all = ImageFolder('Ayana_Bharti/train_mnist_all', transform=data_transform)

# Organize dataset indices by class
class_indices = defaultdict(list)
for idx, (_, label) in enumerate(dataset_all):
    class_indices[label].append(idx)

# Shuffle indices within each class
for label in class_indices:
    random.shuffle(class_indices[label])

# Split indices for each class into training and testing sets
train_indices_client1 = []
train_indices_client2 = []
test_indices = []

for label, indices in class_indices.items():
    total_count = len(indices)
    train_count = total_count * 2 // 3  # Reserve 2/3 of data for training
    client1_count = train_count // 2   # Split training data equally between clients
    client2_count = train_count - client1_count
    test_count = total_count - train_count

    train_indices_client1.extend(indices[:client1_count])
    train_indices_client2.extend(indices[client1_count:client1_count + client2_count])
    test_indices.extend(indices[client1_count + client2_count:])

# Create Subsets for each client and test set
subset_client1 = Subset(dataset_all, train_indices_client1)
subset_client2 = Subset(dataset_all, train_indices_client2)
subset_test = Subset(dataset_all, test_indices)

# Create DataLoaders for clients and test set
batch_size = 64
loader_client1 = DataLoader(subset_client1, batch_size=batch_size, shuffle=True)
loader_client2 = DataLoader(subset_client2, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(subset_test, batch_size=batch_size, shuffle=True)
```

Figure 4.2.1: IID Data Split

4.2.2 Non-IID Dataset

In a non-IID dataset, the data is partitioned in such a way that each client holds a subset of the data that does not necessarily represent the overall distribution of the entire dataset. This results in skewed or unbalanced data across clients, where some clients may have more data from certain classes while others have less or no data from specific classes. This partitioning more closely resembles real-world scenarios, where data can be naturally heterogeneous and non-uniform across different sources.

For example, in the case of the MNIST dataset, a non-IID distribution could involve splitting the dataset such that some clients only receive images of certain digits, while others might have images predominantly from different digit classes. This creates a scenario where each client has a limited and potentially unbalanced view of the dataset, which can make model training more challenging compared to IID setups.

When using the MLP model in an experiment with a non-IID MNIST dataset, the training process becomes more complex. Since each client's data is not representative of the entire dataset, the model must learn to generalize across diverse distributions. This non-uniformity in the data often leads to slower convergence and can result in model performance that is less robust compared to IID scenarios. Fig. 4.2.2 shows code for Non-IID Data Split

4.2.3 How it affects the Global Accuracy

The distribution of data across clients in federated learning, whether IID or non-IID, significantly influences the global accuracy of the trained model.Fig 3.2.3 In the case of **IID data**, where each client receives a representative and evenly distributed subset of the entire dataset, the model tends to converge faster and more efficiently. The gradients computed by each client are consistent with the global data distribution, leading to smoother updates and better generalization. As a result, in experiments like MNIST with IID partitioning, the model achieves high global accuracy quickly and is able to generalize well across all classes. The accuracy trends demonstrated that, under IID conditions, the MLP model was able to learn effectively with fewer communication rounds, reaching a high level of performance within a shorter training time.

However, in **non-IID data**, where clients may have highly skewed or unbalanced data, the global accuracy can be adversely affected. This non-uniform distribution causes inconsistencies in the gradients calculated by each client, resulting in slower convergence, and sometimes poor performance on certain classes that are underrepresented in the local datasets. The global model struggles to generalize effectively since the data is not representative of the entire dataset across all clients. For example, in a non-IID MNIST experiment, where some clients only have data from a subset of digits, the global model may fail to learn features for digits not represented in certain clients' data. Thus, non-IID data requires more communication rounds and careful tuning to achieve acceptable global accuracy, and the model may take longer to converge compared to an IID scenario. Figure 4.2.3

```python
# Split dataset for non-IID simulation
def split_dataset_for_non_iid(class_indices, val_ratio=0.2):
    """
    Splits the dataset into non-IID training and validation indices for clients.

    Args:
        class_indices: Dictionary of class label -> list of sample indices.
        val_ratio: Proportion of samples to reserve for validation.

    Returns:
        Tuple of train and validation indices for each client.
    """
    train_indices_client1, train_indices_client2 = [], []
    val_indices_client1, val_indices_client2 = [], []

    for label in range(10):  # MNIST classes: 0-9
        class_samples = class_indices[label]
        if not class_samples:
            continue  # Skip empty classes
        random.shuffle(class_samples)  # Shuffle to ensure randomness
        class_split = int(val_ratio * len(class_samples))
        class_split = max(1, class_split)  # Ensures at least 1 sample for validation if possible

        if label in [0, 1, 2]:  # Client 1 classes
            val_indices_client1.extend(class_samples[:class_split])
            train_indices_client1.extend(class_samples[class_split:])
        elif label in [3, 4, 5]:  # Client 2 classes
            val_indices_client2.extend(class_samples[:class_split])
            train_indices_client2.extend(class_samples[class_split:])
        else:
            pass  # Currently, these classes are excluded from both clients

    return train_indices_client1, train_indices_client2, val_indices_client1, val_indices_client2
```

Figure 4.2.2: Non-IID Data Split

4.3 Varying Number of Local Training Epochs

4.3.1 Introduction to Epochs Variations

In federated learning, the concept of epochs plays a critical role in determining the performance and convergence of the global model. An epoch refers to one complete pass through the training data on a client's local model. Variations in the number of local training epochs (E) directly influence the trade-off between computational cost and the accuracy of the global model. Increasing E allows the client to perform more training on its local dataset before sending the updated model parameters to the central server. However, a higher number of epochs may lead to overfitting on local datasets, especially in scenarios with non-IID data. Conversely, fewer epochs may result in insufficient training at the client level, delaying the convergence of the global model. Analyzing and optimizing epoch variations is essential for achieving a balance between local training efficiency and global model performance in FL.

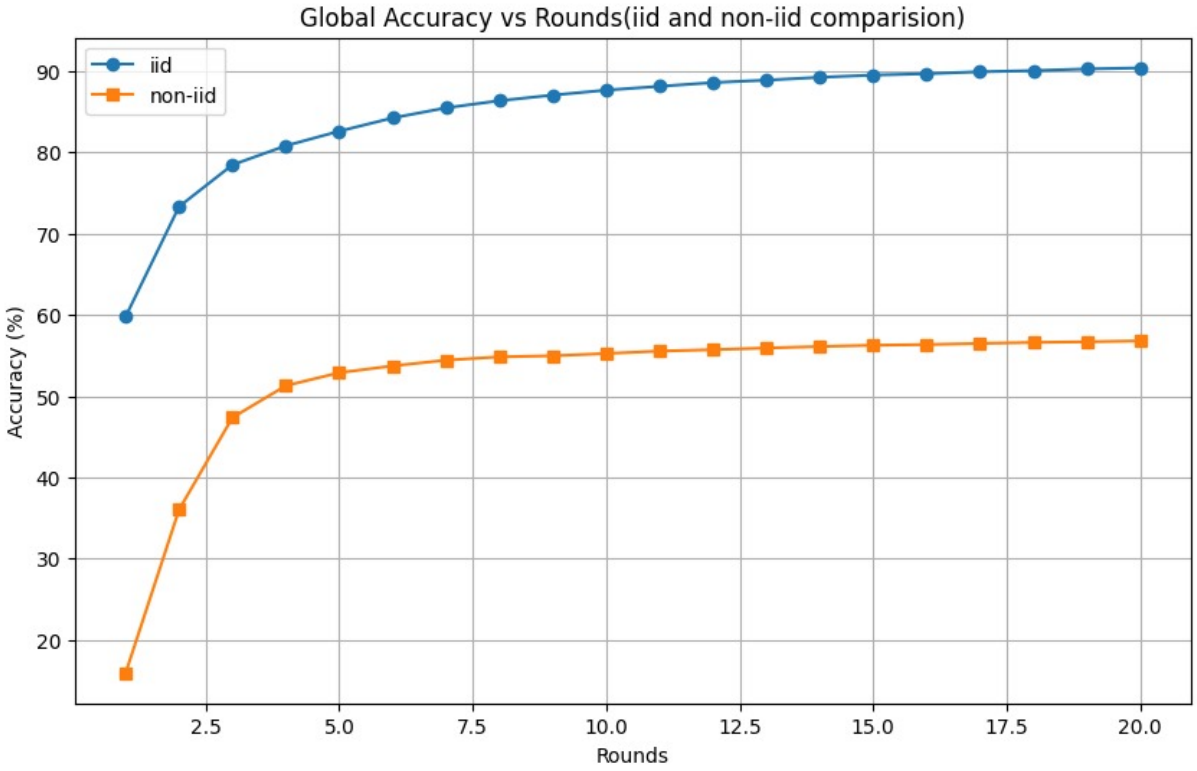

Figure 4.2.3: IID vs Non-IID

4.3.2 Impact on Client's Performance on Varying Epochs

Impact of Increasing Epochs on IID Data

When data is independently and identically distributed across clients, increasing the number of epochs generally enhances model performance. Each client trains on data that represents the global distribution, allowing the local updates to contribute effectively to the global model.

- Low Epochs (E=2): Frequent communication and aggregation ensure consistent global progress but may slow convergence as updates are less refined.
- High Epochs (E=4): More extensive local training improves the quality of updates, leading to faster convergence and better global performance. However, it increases computational cost and risks overfitting, especially if the global model is updated less frequently.

In IID settings, the convergence speed is easier to balance since the client updates are more homogeneous as you increase the epochs, covergence is achieved faster. Figure 4.3.4

Impact of Increasing Epochs on Non-IID Data

Non-IID data introduces heterogeneity, making the impact of increasing epochs more complex. Each client's dataset may contain samples from different distributions, leading to updates that

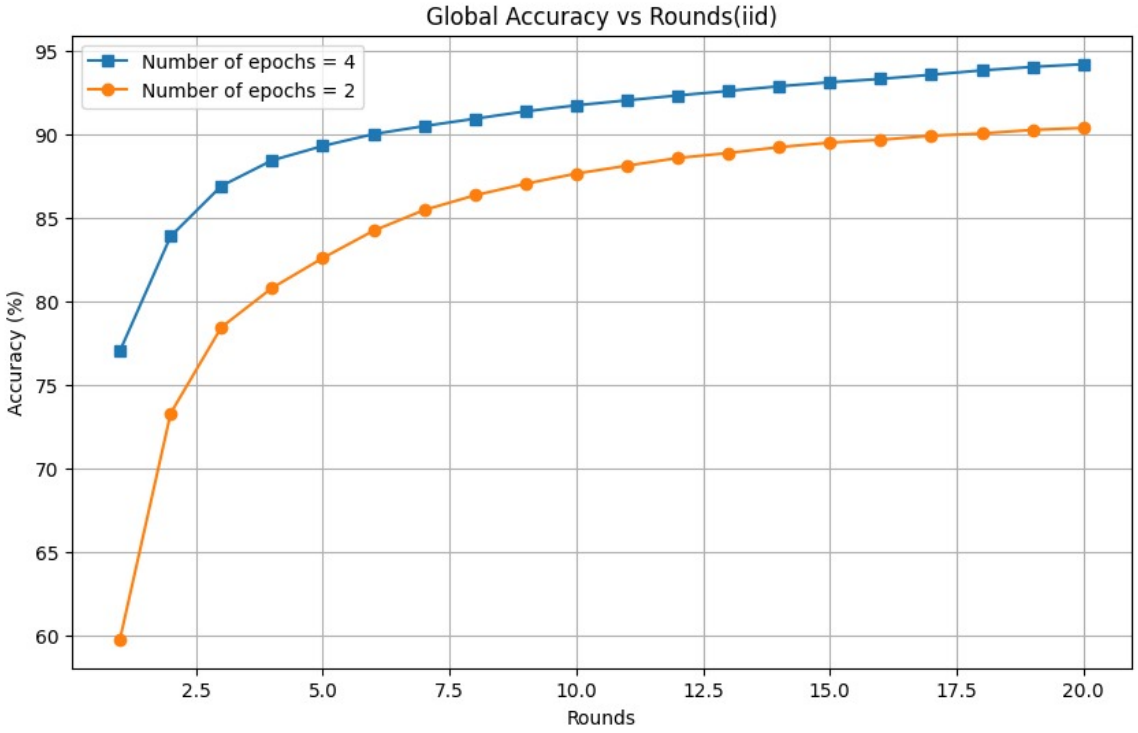

Figure 4.3.4

are less representative of the overall global model.

- Low Epochs (E=2): Frequent aggregation mitigates the effects of local data heterogeneity, as the central server quickly balances disparate updates from clients. However, slower convergence may occur due to limited local training.
- High Epochs (E=4): While longer local training can accelerate convergence in IID data, in non-IID settings, it can exacerbate discrepancies in model updates. Clients overfit their local distributions, leading to divergence in the global model and delayed or suboptimal convergence. Figure 4.3.5

4.3.3 Impact of varying local training epochs on the global model's performance in FL

- Scenario 1 (All Clients Perform 4 Epochs):
  - What Happens: All three clients (A, B, and C) operate with the same number of epochs (4), producing updates that are consistent in refinement and contribution strength.
  - Why It Happens:
    - When all clients perform the same number of epochs, the updates are uniform in quality and frequency.

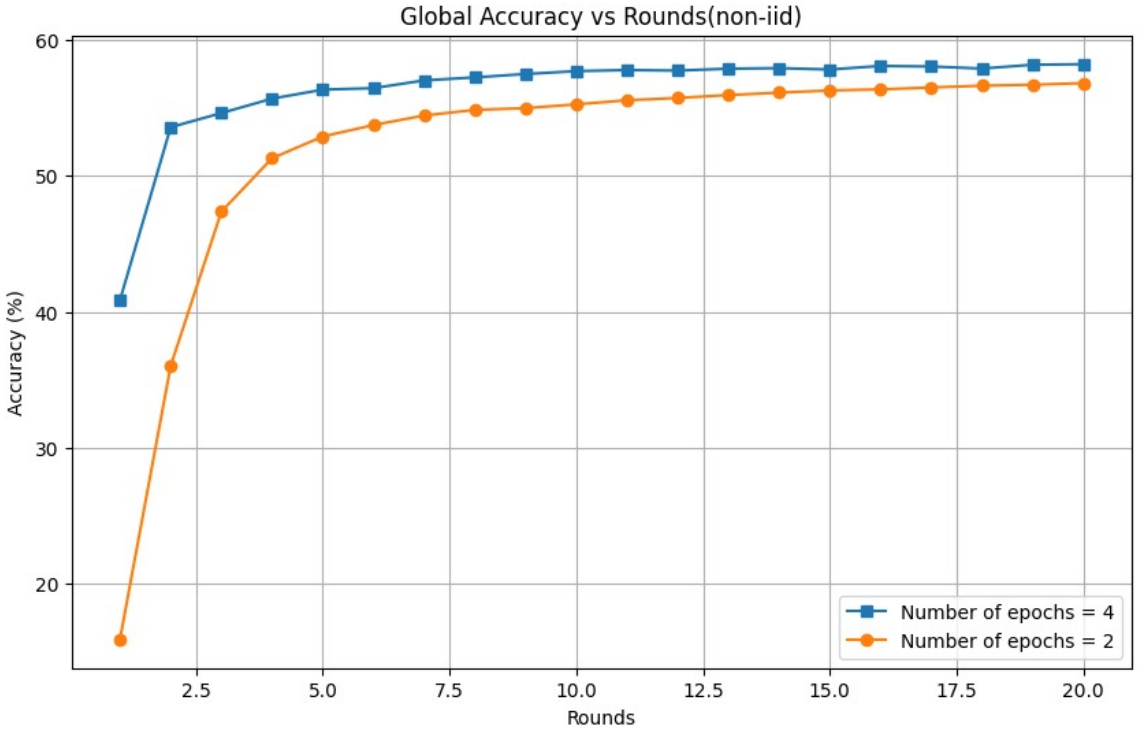

Figure 4.3.5

  - ∗ This balanced contribution ensures that the global model can integrate updates effectively without over-representing or under-representing any client's data.
  - Result: Smooth global convergence and the highest global test accuracy.

- Scenario 2 (Two Clients Complete 4 Epochs, One Does Not):
  - What Happens: Two clients complete 4 epochs, contributing refined updates, while the third client performs fewer epochs, producing less-refined updates.
  - Why It Happens:
    - ∗ The client with fewer epochs generates updates that are less representative of its local data, potentially introducing bias into the global aggregation process.
    - ∗ This creates a moderate imbalance, as the global model gives unequal weight to updates of varying refinement.
  - Result: The global test accuracy is lower than Scenario 1, and convergence slows slightly due to the inconsistency in update quality.

- Scenario 3 (One Client Completes 4 Epochs, Two Do Not):
  - What Happens: Only one client performs 4 epochs, contributing refined updates, while the other two clients provide less-refined updates.
  - Why It Happens:

* The majority of updates come from clients with fewer epochs, which are less representative of their data.
* The refined updates from the single client may dominate the aggregation, leading to overfitting to its data distribution, especially in non-IID settings.
* Additionally, the imbalance in update quality disrupts the global learning process, causing instability and slower convergence.

– Result: The global model achieves the lowest test accuracy. Figure 4.3.6

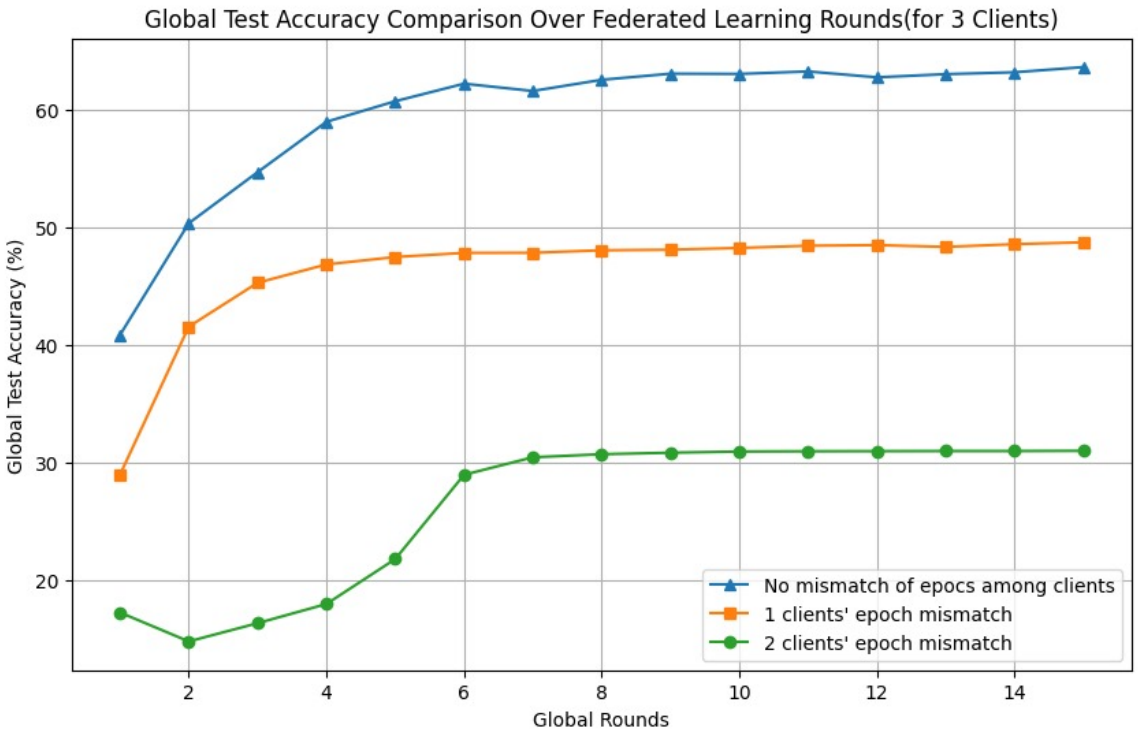

Figure 4.3.6

Github Repo Link: https://github.com/ayanaiiitd/FL-project

Chapter 5

FedProx

5.1 Understanding FedProx

FedProx is an improved version of the basic federated learning method that helps deal with differences in data and device performance across users. In real-world situations, different devices often have different types of data and may not be equally powerful in terms of computing, battery, or network. Unlike the earlier method where all devices had to do the same amount of work, FedProx allows each device to contribute based on its capacity. It also makes sure that the changes made by each device are not too different from the global model, helping the training process stay stable. This makes FedProx more effective and reliable, especially when data is unevenly distributed or devices are not equally capable.

5.2 Proposed Framework: FedProx

The proposed framework, FedProx, is an extension of the standard Federated Averaging (FedAvg) algorithm, designed to handle the challenges posed by system heterogeneity in federated learning environments. In FedAvg, each selected device performs a fixed number of local updates and sends the result to the central server for aggregation. However, this can be problematic in practical settings where devices have different computational capabilities, battery levels, or network conditions. Devices that are slower or less powerful may fail to complete the required computation in time, leading to instability or dropouts.

FedProx addresses this issue in two important ways:

- **Partial Work Tolerance:** FedProx allows devices to perform variable amounts of local computation rather than enforcing a fixed number of local updates for all. This flexibility ensures that slower devices can still participate in training without needing to meet strict uniform requirements. As a result, the system becomes more robust, with fewer dropouts and better overall utilization of available devices.

- **Proximal Term in the Objective Function:** To stabilize training across devices that perform different amounts of work or have diverse data distributions, FedProx adds a proximal term to the local optimization problem. Each device minimizes a modified objective:

$$localobjective = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$$

where $F_k(w)$ is the original loss on the local data of the device, w^t is the global model sent from the server, and μ is a regularization parameter. This proximal term penalizes updates that stray too far from the global model, helping to ensure that local updates stay close to a shared solution and reducing the risk of divergence due to heterogeneity.

By incorporating these changes, FedProx achieves more stable and efficient convergence compared to FedAvg, especially in real-world scenarios where devices are diverse and datasets are non-identically distributed. It also simplifies theoretical analysis and makes it easier to integrate into existing federated learning platforms. In fact, FedAvg can be seen as a special case of FedProx when the regularization parameter mu is set to zero, showing that FedProx is a more general and adaptable approach.

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 5.2.1: FedProx Algo

5.3 Statistical Heterogeneity: Proximal Term

Statistical heterogeneity in federated learning refers to the non-IID nature of data across clients, where each client's local dataset follows a different distribution. This mismatch leads to a divergence in local model updates, which hinders global convergence when aggregating these updates, particularly under methods like `FedAvg`. As statistical heterogeneity increases—from IID to increasingly skewed data distributions—`FedAvg`'s performance degrades significantly, highlighting its instability in such settings.

To address this, the `FedProx` algorithm introduces a proximal term scaled by a penalty parameter μ, which constrains each client's local updates to remain closer to the global model. This modification stabilizes training by mitigating the impact of divergent updates caused by local data variability. While setting $\mu > 0$ might slow convergence under IID data, it proves especially effective in heterogeneous settings, improving robustness and convergence.

Empirical results demonstrate that `FedProx` with an appropriately chosen μ consistently outperforms `FedAvg`, particularly in highly heterogeneous environments. For example, with 90% stragglers, `FedProx` achieves up to 22% higher accuracy compared to `FedAvg`. The choice of μ is crucial and can be adapted dynamically during training—for instance, increasing μ when the loss increases and decreasing it when the loss decreases. This simple adaptive tuning ensures stability without manual intervention.

Additionally, dissimilarity metrics, such as the variance of gradients across clients, increase with data heterogeneity and correlate with worsened convergence. By reducing this dissimilarity through the proximal term, `FedProx` facilitates more stable and effective federated optimization in the presence of statistical heterogeneity.

5.4 Conclusion

In this work, we have proposed `FedProx`, an optimization framework that addresses both systems and statistical heterogeneity inherent in federated networks. `FedProx` enables variable amounts of local computation across devices and incorporates a proximal term to enhance the stability of training. We provide convergence guarantees for `FedProx` under a device dissimilarity assumption, making it suitable for practical federated learning scenarios, including those with stragglers. Our empirical evaluation across a variety of federated datasets supports the theoretical analysis and demonstrates that the `FedProx` framework can significantly improve convergence behavior in realistic, heterogeneous network settings.

Bibliography

[1] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. *arXiv preprint arXiv:1602.05629*, 2016. https://doi.org/10.48550/arXiv.1602.05629.

[2] Subrato Bharati, M. Rubaiyat Hossain Mondal, Prajoy Podder, and V. B. Surya Prasath. Federated Learning: Applications, Challenges and Future Directions. *arXiv preprint arXiv:2205.09513*, submitted on 18 May 2022 (v1), last revised 24 Jun 2022 (this version, v2). https://doi.org/10.48550/arXiv.2205.09513.

[3] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated Optimization in Heterogeneous Networks. *arXiv preprint arXiv:1812.06127*, submitted on 14 Dec 2018 (v1), last revised 21 Apr 2020 (this version, v5). https://doi.org/10.48550/arXiv.1812.06127.