Final Project Report

Aditya Varun V
Indian Institute of Technology, Hyderabad
Kandi, Sangareddy
ai22btech11001@iith.ac.in

Arsh Arora
Indian Institute of Technology, Hyderabad
Kandi, Sangareddy
bm22btech11004@iith.ac.in

Surya Saketh Chakka
Indian Institute of Technology, Hyderabad
Kandi, Sangareddy
ai22btech11005@iith.ac.in

Mayank Parasramka
Indian Institute of Technology, Hyderabad
Kandi, Sangareddy
ai22btech11018@iith.ac.in

Saketh Ram Kumar Dondapati
Indian Institute of Technology, Hyderabad
Kandi, Sangareddy
ai22btech11023@iith.ac.in

Abstract

In this final project report, we present improvements to the ResNet and CNN base models on the PTB-XL database, detailing the methods used to achieve these enhancements. We subsequently explore various federated learning architectures, including the FedAvg, FedProxy algorithm and the Decentralized Federated Learning (DFL) framework. The report also investigates the performance of the FedAvg algorithm, testing it in both Centralized Federated Learning (CFL) and DFL settings. The report also examines the challenges posed by non-IID data across clients, showing poor performance in such cases and discussing strategies to address these issues. We provide insights into achieving faster convergence and improved personalization using FedAvg and Reptile algorithms, supported by detailed explanations and results. Finally, we summarize the different aspects of the overall project, highlighting the key aspects that were achieved. The source code for all experiments can be accessed via the following GitHub repository: GitHub Repository

1. Training and Evaluating over PTBXL

Before entering into a federated setup, we first being to develop and reimplement the state of the art setups for training PTBXL via ResNet, CNN and Vision Transformers. We performed the follwoing tasks as suggested in the PTBXL Documentation to load and pre-process the signals.

1. **Load PTBXL Dataset**:
   - Define the dataset path and load the primary `CSV` files using `pandas`.
   - Organize diagnostic labels by mapping them to the superclass diagnostic category
2. **Pre-processing Signals**:
   - Parse waveform data from the dataset using `wfdb`.
   - Normalize signal amplitudes and resample them to a fixed frequency.
   - Filter signals to remove noise and artifacts, ensuring compatibility with downstream models.
3. **Prepare Labels for Training**:
   - Use a multi-label binarizer to encode diagnostic labels for classification.
   - Handle class imbalance by analyzing label distributions and applying sampling techniques if needed.

After this we get a code ready for being fed into various DL models. The architecture for the DL models are as follows -

- **Convolutional Neural Network (CNN)**
  - **Structure Overview:** The CNN model comprises four 1D convolutional layers with filter sizes of 64, 128, 256, and 512, and kernel sizes of 7, 5, 3, and 3 respectively. Each convolutional layer is followed by batch normalization, ReLU activation, and max-pooling. A global average pooling layer is then applied, followed by a fully connected layer with 1024 units and ReLU activation, a dropout layer with a rate of 0.5, and a final output layer with sigmoid activation.
  - **Number of Parameters:** The CNN model contains ap-

proximately 1,071,370 trainable parameters.

- **Residual Network (ResNet)**
  - **Structure Overview:** The ResNet model starts with a 1D convolutional layer of 64 filters and kernel size 7 with a stride of 2, followed by batch normalization, ReLU activation, and max-pooling. It incorporates four stages of residual blocks with filter sizes of 64, 128, 256, and 512, containing 3, 4, 6, and 3 blocks respectively. Each residual block includes two convolutional layers with batch normalization and ReLU activation, along with skip connections. The model concludes with a global average pooling layer and an output layer with sigmoid activation.
  - **Number of Parameters:** The ResNet model has approximately 7,229,322 trainable parameters.
- **Vision Transformer (ViT)**
  - **Structure Overview:** The ViT model reshapes the input into patches and projects them into a lower-dimensional space using a dense layer. Positional embeddings are added to these projections. The model comprises 8 transformer encoder layers, each with multi-head attention and feed-forward neural networks, employing residual connections and layer normalization. After flattening and applying dropout, the output is fed into a final dense layer with sigmoid activation.
  - **Number of Parameters:** The ViT model consists of approximately 338,018 trainable parameters.

After training on these models for 25 epochs on an GeForce RTX 4080, we received the following results -

Model	F1 Score
CNN	0.73
ResNet	0.73
ViT	0.67

Table 1. F1 Scores for Different Models

these results are comparable with the baselines results as established by the state of the art models as given below -

Model Type	F1 Score
ResNet (Helme et al. 2019)	0.823
CNN (Riberio et al. 2020)	0.81
ViT (Zhou et al. 2024)	0.7

Table 2. F1 Scores for Different Model Types

2. FedAvg

Building upon the baseline models and the improvements achieved by ResNet and CNN on the PTB-XL database, we aimed to replicate these results in a more realistic setting where data is not centralized. Instead, it is distributed across multiple health centers, each holding its own data due to patient privacy concerns. Even if it were possible to aggregate the data centrally, it would be unrealistic for a single health center to manage the entire computational load, as they may lack the necessary resources and infrastructure. For these reasons, we explored Federated Learning (FL) algorithms, starting with *FedAvg*. FedAvg combines local training on clients' data with model aggregation at a central server. The core idea of FedAvg is to allow clients to train locally on their data, then aggregate their local updates to improve the global model.

2.1. Algorithm and Implementation

In implementing Federated Learning (FL), we had two options: a **Centralized Federated Learning (CFL)** approach, where a central server is responsible for aggregating model updates after each epoch, or a **Decentralized Federated Learning (DFL)** approach, where the central server is replaced by peer-to-peer communication among clients. In DFL, clients directly exchange model updates with each other to perform aggregation, eliminating the need for a central coordinator.

2.1.1 Centralised Federated Learning

Algorithm 1 Federated Averaging (FedAvg) Algorithm

```
0: Initialization:
0: The central server initializes the global model parameters $w(0)$ (random initialization).
0: for each round $t = 1, 2, \ldots, T$ do
0:     Client Training:
0:     for each client $i = 1, 2, \ldots, N$ do
0:         Client $i$ trains the model on its local dataset $D_i$ for a predefined number of local epochs.
0:         Let $w_i(t)$ represent the model weights of client $i$ after the $t$-th round of training.
0:     end for
0:     Model Aggregation:
0:     Each client sends its updated model parameters $w_i(t)$ to the central server.
0:     The central server computes the weighted average of the local model updates:
           $$w(t+1) = \frac{1}{N} \sum_{i=1}^{N} w_i(t)$$
0:     Update the global model: $w(t+1)$ becomes the new global model.
0:     Iteration: The process is repeated for the next round $t+1$.
0: end for=0
```

2.1.2 Testing and Results

We ran the algorithm with 5 clients, 10 rounds, and each client trained for 10 epochs. The code was tested on both the CNN and ResNet architectures, with either sampling 3 clients per round or using all 5 clients in each round.

Model	Sample	Training Accuracy	Testing Accuracy
CNN	3	0.8759	0.5850
CNN	5	0.8804	0.5889
ResNet	3	0.8244	0.6601
ResNet	5	0.8661	0.6719

Table 3. CFL Model Training and Validation Accuracy for Different Client Numbers

2.1.3 Decentralised Federated Learning

In **Decentralized Federated Learning (DFL)**, clients exchange model updates directly with each other, bypassing the need for a central server. This approach distributes the computational load, improves scalability, and eliminates the single point of failure. However, it introduces challenges such as coordination and communication overhead between clients.

Topology refers to the network structure that defines how clients are connected and how they communicate with each other. In our implementation, we explored two types of topologies:

- **Fully Connected (FC)**: Every client is connected to every other client, allowing for direct communication and model updates between all clients.
- **Ring**: Clients are arranged in a circular structure, where each client communicates only with its immediate neighbor.

2.1.4 Testing and Results

The algorithm was tested using a CNN model with similar settings as the CFL scenario: 5 clients, 10 rounds, and each client trained for 10 epochs. The results for both topologies are shown in Table 4.

Topology	Training Accuracy	Testing Accuracy
Fully Connected	0.8804	0.5889
Ring	0.8150	0.5629

Table 4. DFL Training and Validation Accuracy for Different Topologies

2.2. Conclusions

From Table 3 and Table 4, we observe the following:

1. Although we were not able to achieve results identical to the baseline models, the accuracy of our Federated Learning (FL) model steadily increased and started to converge towards the baseline model's performance.
2. As expected, the Fully Connected Decentralized Federated Learning (DFL) model and the Centralized Federated Learning (CFL) model, both with the same weight initialization, showed identical accuracy.
3. Increasing the number of clients in each round led to improved accuracy, as observed during the 10-epoch training. However, as the model trains for more epochs, the accuracies are expected to converge to similar values.
4. The Fully Connected topology showed a similar improvement, converging faster due to more frequent and comprehensive exchange of information between clients in each epoch.

3. FedAvg over Non-IID Data

One key challenge in FL is the non-IID (non-Independent and Identically Distributed) data scenario, where each client has data distributions that differ from each other. Here we demonstrate how Federated Averaging (FedAvg) algorithm performs poorly when the data across clients is non-IID.

3.1. Synthetic Data Generation

Non-IID Data for multiple clients is generated as follows:

- The data for each client is simulated using bivariate Gaussian distributions with different means and covariances.
- The data is divided into two classes, and each client receives a separate dataset for training and testing.
- The datasets are then shuffled for randomness, and the training and testing data for each client are stored separately.

The data for each client is then visualized using scatter plots.

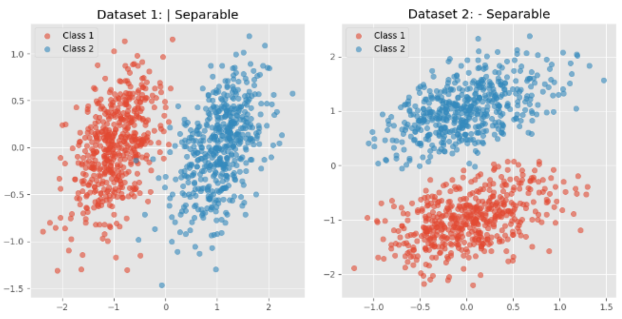


Figure 1. Generated datasets for multiple clients. Each subplot shows the data distribution for a different client.

3.2. Training and Evaluation

The model used in the experiment is a simple logistic regression model with a binary cross-entropy loss function.

The FedAvg algorithm was run for 10 rounds, and 3 clients were chosen randomly for each round. The global model is updated by averaging the weights of the models trained by the selected clients. The training accuracy of each client is recorded for each round, and plots are generated to show how the training accuracy changes over the rounds for each client using the global weights.

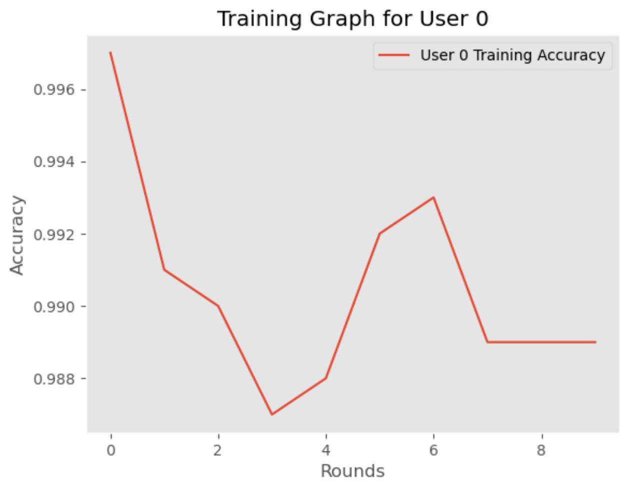


Figure 2. Training accuracy for client 0 over multiple rounds.

3.3. Results and Observations

The Training accuracy on the entire training data for all clients is 82.44% and testing accuracy is 66.01%. The main observation in this experiment is that the global model, after being trained using the FedAvg algorithm on non-IID data, does not generalize well when evaluated on individual clients' data. This is due to the mismatch between the local data distributions and the global averaged model, leading to poor performance on clients whose data differs significantly from the global model. The number of times each client is selected during the federated learning rounds is also tracked. The frequency of participation affects the updates to the global model, but the non-IID nature of the data remains the primary factor in poor performance.

4. FedProx

FedAvg [3], is a popular algorithm to perform FL, where each client updates received weights locally, using E epochs of SGD on K clients. The updated weights are passed to a central aggregator, to finally combine these updated weights. While this will work wonderfully when the clients have IID datasets, we can see issues when we have heterogenous data. It can be shown to diverge/oscillate when the data is distributed non-IID. This happens, since each client might end up trying to optimize their own loss

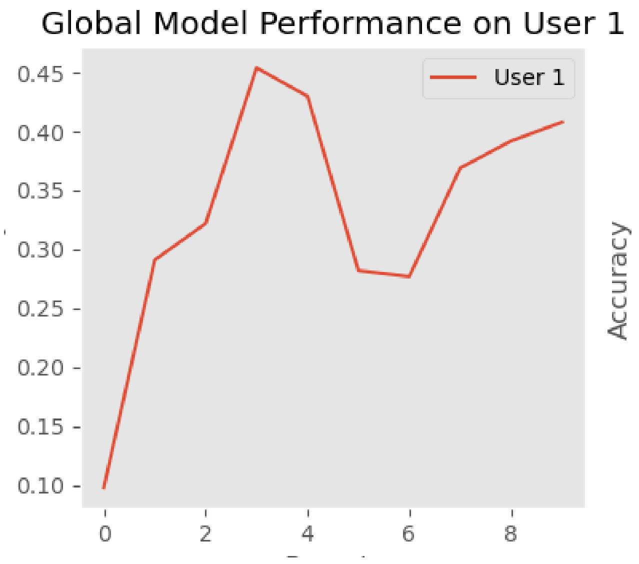


Figure 3. Poor global model accuracy for client 1 over multiple rounds.

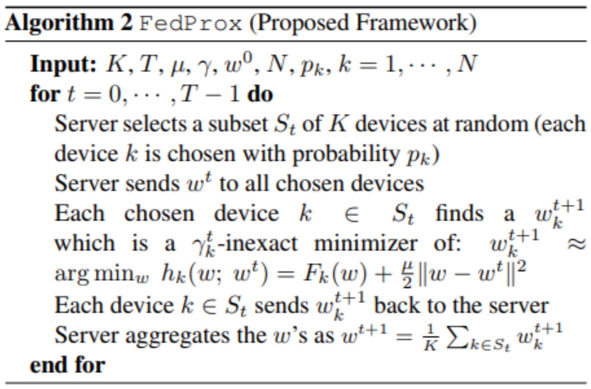
function. Furthermore, $FedAvg$ forces the clients to perform E epochs every round. This is unreasonable since each client need not have the same computational resources, and some clients may get dropped for not finishing E epochs.

[2] $FedProx$ offers a solution to both of these problems! They introduce a term γ_k^t, which is a measure of how much local computation needs to happen at client k at the t-th iteration for its work to be considered by the aggregator. That is, it simply needs to compute up until the gradient improves γ_k^t times the initial gradient for the current loss function. This can be high, for lower-end clients and lower for those with better resources. Note that a lower γ_k^t corresponds to more work, since the gradient must decrease more. The paper terms this **"Tolerating partial work"**. Furthermore, a **proximal term** $\frac{\mu}{2}||w - w_t||^2$ is added to the cost function at each client. This is so that each updated weight is closer to the current global weight. This stops clients from trying to minimize their local cost function only. This helps the algorithm perform better in non-IID situations as compared to *FedAvg*.

4.1. Testing and Results

For the implementation, we used the code from here as a base. Since we have the same computational resources for each client, we decided to forgo the γ_k^t term, and only implement the proximal term.

We use the CNN and MLP architecture directly from the repository, and on non-IID distributed MNIST data. Number of clients $K = 100$, number of sampled clients $M = 10$ and $\mu = 0.1$ in our experiments. Training was



```
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 $\gamma_k^t$-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 4. The *FedProx* algorithm, as described in [2]

completed for 10 global rounds. The chosen optimizer was SGD(stochastic gradient descent).

The MNIST data was distributed non-IID by splitting the data in 200 shards, where each shard has similar labelled data. Then, each client is given 2 shards. This way, each client ends up biased towards a small subset of the dataset.

The repository above obtains the following results descibed in Table 5 on the MNIST dataset for IID and non IID distributions, using SGD, using *fedavg*.

Model	IID	Non-IID
mlp	88.38%	73.49%
cnn	97.28%	75.94%

Table 5. Test accuracy for MNIST using CNN and MLP in iid and non iid settings

The results are summarized in Table 6.

Model	Training Accuracy	Testing Accuracy
mlp	79.24%	69.63%
mlp_fedprox	88.33%	77.78%
cnn	90.55%	83.09%
cnn_fedprox	88.77%	77.66%

Table 6. Training and Testing accuracy

4.2. Conclusions

From Table 6, we observe the following:

1. For the MLP, we see an increasing of almost 10% in training and testing accuracies.
2. For the CNN, unfortunately, we see a drop in 5%, in the testing accuracies.

We conclude see that *FedProx* can improve results for heterogenous data distributions. Although we weren't able to achieve the 20% improvement as [2] boasts, we still see considerably better results in the MLP network.

5. Personalized FedAvg

This algorithm was tested to improve performance when clients have non-IID data in the federated learning setting. The following section is based on this research paper: [1]

5.1. Algorithm

The Personalized FedAvg algorithm combines the strengths of FedAvg and Reptile to improve convergence, stability, and personalization in non-IID settings. The algorithm steps are as follows:

1. **Global Initialization:** Run FedAvg with momentum SGD as the server optimizer, using a relatively larger number of local epochs (E).
2. **Fine-Tuning:** Switch to Reptile with Adam as the server optimizer for fine-tuning, using a small number of local steps (K).
3. **Personalization:** Train the obtained global model locally for each client to personalize for the client (P).

5.1.1 Reptile Algorithm

```
Algorithm 2 Reptile (Serial Version)
0: Initialize: $\phi$, the vector of initial parameters
0: for iteration = $1, 2, \dots$ do
0:   Sample task $\tau$, corresponding to loss $L_\tau$ on weight vectors $\phi_e$
0:   Compute $\phi_e \leftarrow U_\tau^k(\phi)$, denoting $k$ steps of SGD or Adam
0:   Update $\phi \leftarrow \phi + \epsilon(\phi_e - \phi)$
0: end for=0
```

We see that the reptile algorithm is similar to the FedAvg algorithm, where the main difference lies in the update step. This algorithm is used to fine-tune the global model, after the FedAvg steps [4]

5.2. Motivation for Applying the Algorithm

The Personalized FedAvg algorithm is designed to address challenges in federated learning with non-IID data by:

- Ensuring **faster convergence** with FedAvg during the initial training phase.
- **Stabilizing** the global model through Reptile-based fine-tuning with Adam.
- Leveraging **personalized adaptation** to align models with client-specific data distributions.

5.3. Testing and Results

We ran the personalized FedAvg algorithm on the non-IID Synthetic Gaussian clients, for $E = 10$ local epochs, $K = 10$ clients per round, and $P = 10$ personalization epochs. The results are summarized in Figure 6 and Table 7.

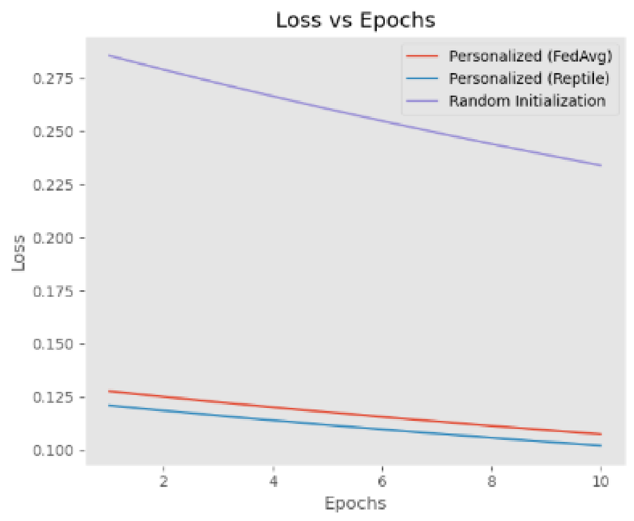


Figure 5. Loss vs. Epoch During Personalization (Client 0)

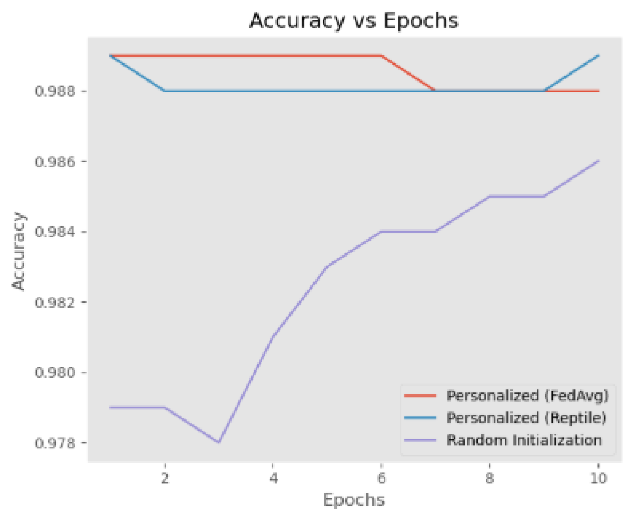


Figure 6. Accuracy vs. Epoch During Personalization (Client 0)

Model	Average Loss	Average Accuracy
m_per_fed_avg	0.3798	0.8302
m_per_reptile	0.3770	0.8328
m_random	0.9505	0.3792
global_fed_avg	0.3945	0.8275
global_reptile	0.3905	0.8292

Table 7. Average Loss and Accuracy for Different Models, averaged over the clients.

5.4. Conclusions

From Figure 6 and Table 7, we observe the following:

1. Using the FedAvg/PerFedAvg algorithm, we are able to adapt to the client much quicker (fewer epochs) compared to the randomly initialized weights (Figure 6).
2. Personalized models achieve better performance compared to the global models on the clients (Table 7).
3. Applying the Reptile algorithm after the FedAvg algorithm results in marginally better performance, likely due to fine-tuning by the Reptile method (Table 7).

6. Final Conclusion

The report systematically explores advancements in training models such as ResNet, CNN, and Vision Transformers for the PTB-XL dataset, followed by implementing Federated Learning (FL) frameworks like FedAvg, FedProx, and Personalized FedAvg. We address challenges in non-IID data settings, demonstrating that personalized algorithms and fine-tuning strategies improve performance. Additionally, comparisons with baseline methods validate the efficacy of their approaches, particularly in adapting to client-specific distributions. Exploring decentralized FL frameworks adds depth by tackling practical data privacy concerns in distributed systems.

In conclusion, the project effectively combines traditional deep learning models with innovative FL methods to bridge gaps in distributed training. Incorporating strategies for handling heterogeneous data distributions underscores the relevance of this research to real-world applications, particularly in healthcare. By exploring both centralized and decentralized federated methods, this project lays a strong groundwork for further investigations into scalable and privacy-preserving learning frameworks.

References

[1] Yihan Jiang, Jakub Konečný, Keith Rush, and Sreeram Kannan. Improving federated learning personalization via model agnostic meta learning, 2023. 5

[2] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks, 2020. 4, 5

[3] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2023. 4

[4] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms, 2018. 5