Demonstration of Federated Learning using multi-systems

Artifial neural network model

1st Rahul Mahto
Department of information technology
Indian Institute of Information Technology, Allahabad
Prayagraj, India
Roll No: IIT2020022
Email: iit2020022@iiita.ac.in

2nd Shubham Kumar
Department of information technology
Indian Institute of Information Technology, Allahabad
Prayagraj, India
Roll No: IIT2020007
Email: iit2020007@iiita.ac.in

3rd Aashish Agrawal
Department of information technology
Indian Institute of Information Technology, Allahabad
Prayagraj, India
Roll No: IIT2020009
Email: iit2020009@iiita.ac.in

4th Rohan Singhvi
Department of information technology
Indian Institute of Information Technology, Allahabad
Prayagraj, India
Roll No: IIB2020004
Email: iib2020004@iiita.ac.in

5th Akash Biswas
Department of information technology
Indian Institute of Information Technology, Allahabad
Prayagraj, India
Roll No: IIT2020001
Email: iit2020001@iiita.ac.in

***Abstract*—Federated learning is a distributed machine learning approach that allows multiple systems to collaboratively train a model without sharing their data. In this project, we demonstrate the use of federated learning to train an artificial neural network (ANN) model using multiple systems. Each system has a dataset and contributes to the training process by computing local gradients and sharing them with a central server. The central server aggregates the gradients and updates the global model, which is then distributed back to the participating systems for further training. We implement this approach using the Tensorflow library, which provides tools for privacy-preserving machine learning and secure multi-party computation. Our experimental results demonstrate that federated learning can effectively train an ANN model using multiple systems while preserving data privacy and security.**

***Keywords*—Federated Learning, Artificial neural network, tensorflow, privacy-preserving, multi-client, server, MNIST.**

I. Introduction

Federated learning is a machine learning technique that enables multiple parties to collaboratively train a machine learning model without sharing their private data. In traditional machine learning, all the data is collected in a centralized location, and the model is trained on this data. However, in federated learning, the data remains on the devices or servers of the parties involved, and the model is trained in a distributed manner. In a federated learning setup, the parties involved (e.g., mobile devices, hospitals, or financial institutions) each have their own private datasets. The parties collaborate to train a global model by sharing only the model parameters and not the raw data. The global model is sent to the devices or servers for local training using their own data. The model updates are then sent back to a central server, where they are aggregated to update the global model. Federated learning is particularly useful in situations where

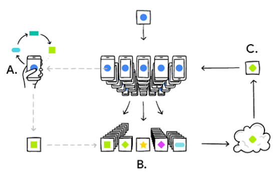


Fig. 1. Federated Learning Illustration

data privacy is a concern, as it enables the training of a robust model while protecting sensitive data. For example, in healthcare, federated learning can be used to train a model on data from multiple hospitals without the need to share patients' sensitive medical information. Similarly, in finance, federated learning can be used to train a model on data

from multiple financial institutions without sharing sensitive financial data. Federated learning also has other advantages over traditional machine learning, such as the ability to train on large datasets distributed across different devices, reducing the need for data transfer and reducing the risk of data breaches.
The MNIST dataset is a popular dataset used for image classification, containing a large number of handwritten digits. The aim of federated learning is to train an ANN model on this dataset in a distributed manner by leveraging the computational power of multiple devices or servers, while ensuring that the individual training data remains private and secure.

II. Objective

The objective of federated learning with Artificial Neural Networks (ANN) using the MNIST dataset is to train a machine learning model across multiple decentralized devices or servers without exchanging their data, while maintaining the privacy and security of the individual data.

A. Improve the accuracy of the model:

By leveraging the diverse and abundant data present on various devices, federated learning can help to improve the accuracy of the ANN model.

B. Preserve data privacy and security:

Federated learning ensures that individual data remains private and secure, as the training data is never transferred to a central location or third-party server.

C. Reduce communication costs:

Federated learning can help to reduce the communication costs associated with centralizing the training data on a single server.

D. Enhance model scalability:

Federated learning can be used to train models on a large scale, by leveraging the computational power of multiple devices or servers.

III. Artificial Neural Network

Artificial Neural Networks (ANNs) are a type of machine learning model inspired by the structure and function of the human brain. ANNs are widely used for various tasks, including classification, regression, and pattern recognition. Here's a high-level explanation of how an ANN model works:

A. Neurons

An ANN is composed of individual units called neurons, which are organized into layers. Each neuron receives input from multiple sources and produces an output based on its internal computation. The output of a neuron is typically passed through an activation function, which introduces non-linearity into the network.

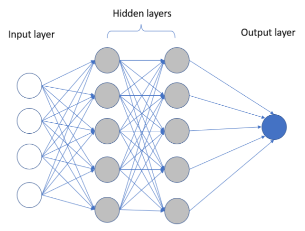


Fig. 2. Neural Network

B. Layers

ANNs are typically structured into layers, consisting of an input layer, one or more hidden layers, and an output layer. The input layer receives the initial data or features, and subsequent layers process and transform the input to produce the final output. Hidden layers, located between the input and output layers, perform the bulk of the computational work in an ANN.

C. Connections and Weights

Neurons within an ANN are connected through weighted connections. Each connection between two neurons has an associated weight, which determines the strength and influence of the connection. The weights are adjusted during the training process to optimize the performance of the network

D. Forward Propagation

During the forward propagation phase, the input data is fed into the network through the input layer. The input data is multiplied by the corresponding weights and passed through the activation function of each neuron in the hidden layers. This process continues layer by layer until the output layer produces the final prediction or output of the network.

E. Loss Function

The output of the network is compared to the expected or target output using a loss function. The loss function measures the dissimilarity between the predicted output and the actual output. The choice of the loss function depends on the task at hand, such as mean squared error (MSE) for regression or categorical cross-entropy for classification

F. Back propagation and Gradient Descent

The goal of training an ANN is to minimize the loss or error between the predicted output and the actual output. Backpropagation is a technique used to calculate the gradient of the loss function with respect to the weights of the network. The gradients are then used in the optimization algorithm, such as gradient descent, to update the weights and reduce the loss.

G. Training

The training process involves iteratively feeding the training data through the network, calculating the loss, and updating the weights using backpropagation and gradient descent. This process continues for multiple epochs until the network learns to make accurate predictions and minimize the loss on the training data.

H. Evaluation and Prediction

Once the ANN is trained, it can be used for making predictions on new, unseen data. The input data is fed into the network through the input layer, and the forward propagation process generates the predicted output.

ANNs can be designed in various architectures, such as feedforward neural networks, convolutional neural networks (CNNs), recurrent neural networks (RNNs), and more. Each architecture has its specific characteristics and is suited for different types of data and tasks

IV. SOCKET PROGRAMMING

Socket programming is a technique that allows communication between two computers over a network using sockets. It enables the exchange of data, messages, or information between client and server applications. Here's an explanation of socket programming:

A. Client-Server Model

- Socket programming follows the client-server model, where one computer acts as a server, listening for incoming connections, and another computer acts as a client, initiating the connection. - The server provides a service or resource that the client requests or interacts with.

B. Sockets

C. Socket APIs FlaskAPI

- Socket programming is typically performed using socket Application Programming Interfaces (APIs). - Socket APIs provide functions and methods that enable the creation, connection, sending, and receiving of data over sockets.

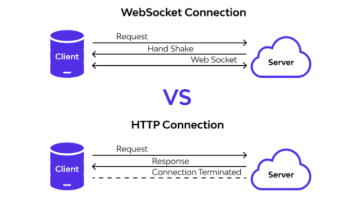


Fig. 3. Communication of client and server

D. Server Side:

- The server creates a socket and binds it to a specific IP address and port number. - The server then listens for incoming connections on that socket. - When a client connects to the server, the server accepts the connection and creates a new socket specifically for that client.

E. Client Side:

- The client creates a socket and connects it to the server's IP address and port number. - Once the connection is established, the client can send data to and receive data from the server.

F. Data Transfer:

- After the connection is established, data can be transferred between the client and server. - The client can send requests or messages to the server, and the server can respond with the requested data or perform actions based on the client's request. - Data is typically sent in chunks or packets, and the client and server take turns sending and receiving data.

G. Protocols:

- Socket programming can be performed using various protocols, such as TCP (Transmission Control Protocol) or UDP (User Datagram Protocol). - TCP provides reliable, ordered, and error-checked communication, suitable for applications where data integrity is crucial. - UDP provides faster and more lightweight communication but does not guarantee reliable delivery or ordering.

H. Closing the Connection:

- Once the communication is complete or the client no longer needs the server's service, either the client or the server can close the connection. - Closing the connection involves releasing the resources allocated for the socket and terminating the communication.

Socket programming is widely used in various networked applications, including web servers, chat applications, file transfer protocols, and more. It provides a flexible and powerful way to establish communication and exchange data between computers over a network.

V. METHODOLOGY

Here is a detailed methodology for the implementation: - Data Preparation: The first step is to download the MNIST dataset and split it into non-overlapping subsets for each client. The repository includes a mnist iid py script that performs this data preparation step. This script generates four pickle files containing the train and test datasets, and their corresponding labels, for each client.

A. Model Architecture:

The implementation uses a simple two-layer neural network model with 784 input units, 128 hidden units, and 10 output units. The model is defined using PyTorch, a popular deep learning framework. The model architecture is specified in the model.py file.

B. Client and Server Implementation:

The repository includes the implementation of both client and server sides. The client code trains a local model on its subset of the MNIST dataset and sends the model updates to the server for aggregation. The server code aggregates the updates and returns the new global model to the clients for further training. The implementation is built using tensorflow, a privacy-preserving machine learning framework that allows secure and private communication between the clients and the server.

C. Training Process:

The training process involves multiple rounds of training on the local datasets followed by aggregation of the model updates at the server. During each round of training, the clients randomly sample a subset of their data, train their local model on this subset, and send the model updates to the server. The server aggregates the updates using a simple averaging method, and sends the new global model back to the clients. The process repeats for a specified number of rounds.

D. Noisy and Malicious Client Simulation:

The implementation also provides functionality to simulate the presence of noisy or malicious clients. The repository includes scripts to generate noisy and malicious client data, which can be used to test the robustness of the system to such attacks.

E. Model Evaluation:

The implementation includes functionality to test the performance of the global model on a separate test dataset, allowing for evaluation of the model's accuracy over time. The repository includes a test.py script that evaluates the final global model on the test dataset.

VI. IMPLEMENTATION

he implementation is based on the client-server architecture commonly used in federated learning. In this architecture, the clients (in this case, individual machines or devices) train their own local models on their respective datasets, and the server aggregates these models to create a global model that can improve its performance over time. This approach allows for distributed training on sensitive or private data without requiring data to be centralized on a single server.

The implementation uses the Ternsorflow framework to build and train the neural network model. The model used in this implementation is a simple two-layer neural network with 784 input units (representing the 28x28 pixel images in the MNIST dataset), 128 hidden units, and 10 output units (representing the 10 possible digit classes). This architecture is relatively simple, but it provides a good starting point for experimenting with federated learning on the MNIST dataset.

The implementation also uses the tensorflow library to implement federated learning. tensorflow is an open-source library that provides a simple and easy-to-use interface for implementing privacy-preserving machine learning techniques, including federated learning. tensorflow enables secure and privacy-preserving communication between the clients and the server, allowing the clients to train their models locally without revealing their sensitive data to the server.

In the implementation, the server.py file starts the server and waits for clients to connect. Once clients have connected, the server initiates the federated learning process by broadcasting the initial global model to all clients. Each client trains the local model using its own data and sends the updated model parameters back to the server.

The server aggregates the model updates using a simple average aggregation method and sends the updated global model back to the clients. This process is repeated for a specified number of rounds, and the global model gradually improves as more data is incorporated into the training process. The client.py file is responsible for connecting to the server, downloading the current global model, and training the local model on the client's data. The client then sends the updated model parameters back to the server for aggregation. The implementation includes functionality to simulate the presence of noisy or malicious clients by randomly dropping or corrupting model updates sent by clients.

The implementation also includes functionality to test the performance of the global model on a separate test dataset. After each round of training, the server.py file calculates the accuracy of the global model on the test dataset and logs the results to the console.

Overall, the implementation provides a simple and easy-to-understand example of federated learning using the MNIST dataset. It demonstrates how PyTorch and tensorflow can be used together to implement a secure and privacy-preserving federated learning system, and it can be used as a starting point for experimenting with more complex models and datasets.

VII. CHALLENGES

The repository you provided contains an implementation of Federated Learning using the MNIST dataset. In Federated Learning, the model training is distributed across multiple clients, allowing data to remain on the clients' devices while the model is trained on a central server. Here are some challenges you may encounter when working with this repository:

1. Data Privacy: Federated Learning aims to protect the privacy of client data by keeping it on their devices. However, ensuring data privacy and security can be challenging, as sensitive information may still be at risk during communication between the clients and the server. It is crucial to implement robust security measures to protect the data during transmission and storage.

2. Communication and Synchronization: In Federated Learning, the server needs to communicate with multiple clients to aggregate their local model updates. Ensuring reliable communication and synchronization between the server and clients can be challenging, especially when dealing with unreliable or slow network connections. Handling communication failures, packet losses, and network latency is essential for efficient Federated Learning.

3. Heterogeneous Data and Device Disparities: The clients in Federated Learning can have diverse datasets and hardware capabilities. It is common for clients to have varying data distributions, data quality, and device performance. These differences can impact the convergence and generalization of the global model. It requires careful handling of heterogeneity, such as adaptive learning rate schedules, model architecture adjustments, and data preprocessing techniques.

4. Model Optimization: Training a global model in Federated Learning involves aggregating local model updates from multiple clients. However, due to the limited communication

and computational resources, the server might have access to only a fraction of the total training data. This limited access can affect the model's performance and generalization. Developing efficient aggregation strategies and optimization algorithms to mitigate these challenges is crucial.

5. Client Heterogeneity and Participation: In a federated setting, clients may have varying levels of participation or reliability. Some clients may have unreliable connections, limited resources, or drop out during the training process. Handling client heterogeneity, such as designing protocols to handle missing or delayed updates, is necessary to maintain the integrity of the federated learning system.

6. Bias and Fairness: Federated Learning can introduce bias if the clients' data distribution is not representative of the overall population. If certain groups of clients are over- or under-represented, the trained model may be biased towards those groups. Ensuring fairness in the federated learning process requires careful consideration of data collection strategies and fairness-aware aggregation techniques.

7. Scalability: As the number of clients and the complexity of the model increase, the scalability of Federated Learning becomes a challenge. Coordinating a large number of clients and managing their resources efficiently can be complex. Optimizing the system for scalability, both in terms of the number of clients and the model size, is important for practical deployment.

These challenges highlight some of the key considerations when working with the Federated Learning repository you provided. Addressing these challenges requires careful design, implementation, and experimentation to achieve effective and robust federated learning systems.

VIII. FUTURE SCOPE

The project "Federated Learning using MNIST Data" provides an excellent starting point for exploring the potential of Federated Learning for training machine learning models on distributed data while preserving privacy. The project has many potential future scopes that could improve its efficiency, scalability, and applicability to real-world scenarios. Some of the potential future scopes are:

A. Scalability:

The current implementation of Federated Learning in the project is limited to a small number of devices, which may not be practical for real-world scenarios. Future work could focus on optimizing Federated Learning algorithms for large-scale distributed systems, where millions of devices are involved.

B. Data Privacy:

The project uses a simple approach to privacy by encrypting the data before sending it to the central server. However, more advanced privacy-preserving techniques such as differential privacy and homomorphic encryption could be explored to improve the privacy of the data.

C. Heterogeneity:

The MNIST dataset used in the project is a standardized dataset, where each image is of the same size and format. However, in real-world scenarios, the data can be highly heterogeneous, making it challenging to train the model. Future work could focus on developing Federated Learning algorithms that are robust to heterogeneous data.

D. Transfer Learning:

Transfer Learning is a powerful technique that can be used to improve the performance of machine learning models by leveraging knowledge learned from related tasks. Future work could focus on exploring the potential of Transfer Learning in Federated Learning scenarios.

E. Model Compression:

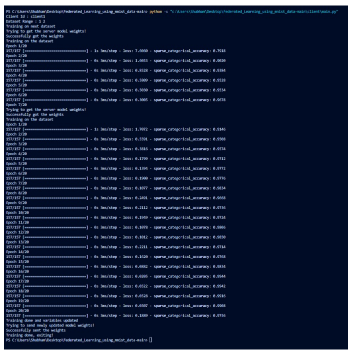
Federated Learning involves training a large number of models on different devices, which can result in a significant amount of redundant information. Model compression techniques could be explored to reduce the model size and improve the efficiency of Federated Learning.

F. Real-world applications:

The project demonstrates the potential of Federated Learning for training machine learning models on distributed data while preserving privacy. Future work could focus on applying Federated Learning to real-world scenarios such as medical diagnosis, financial prediction, and smart city applications, among others

IX. RESULTS

A. client 1



```
PS C:\Users\Shubham\Desktop\Federated_Learning_using_mnist_data-main> python -u "c:\Users\Shubham\Desktop\Federated_Learning_using_mnist_data-main\client\main.py"
Client Id : client1
Dataset Range : 1 2
Training on next dataset
Trying to get the server model weights!
Successfully got the weights
Training on the dataset
Epoch 1/20
157/157 [==============================] - 1s 3ms/step - loss: 7.6060 - sparse_categorical_accuracy: 0.7918
Epoch 2/20
157/157 [==============================] - 0s 3ms/step - loss: 1.6053 - sparse_categorical_accuracy: 0.9020
Epoch 3/20
157/157 [==============================] - 0s 3ms/step - loss: 0.8528 - sparse_categorical_accuracy: 0.9384
Epoch 4/20
157/157 [==============================] - 0s 3ms/step - loss: 0.5809 - sparse_categorical_accuracy: 0.9528
Epoch 5/20
157/157 [==============================] - 0s 3ms/step - loss: 0.5030 - sparse_categorical_accuracy: 0.9534
Epoch 6/20
157/157 [==============================] - 0s 3ms/step - loss: 0.3005 - sparse_categorical_accuracy: 0.9678
Epoch 7/20
Trying to get the server model weights!
Successfully got the weights
Training on the dataset
Epoch 1/20
157/157 [==============================] - 1s 3ms/step - loss: 1.7072 - sparse_categorical_accuracy: 0.9146
Epoch 2/20
157/157 [==============================] - 0s 3ms/step - loss: 0.5591 - sparse_categorical_accuracy: 0.9508
Epoch 3/20
157/157 [==============================] - 0s 3ms/step - loss: 0.3816 - sparse_categorical_accuracy: 0.9574
Epoch 4/20
157/157 [==============================] - 0s 3ms/step - loss: 0.1799 - sparse_categorical_accuracy: 0.9712
Epoch 5/20
157/157 [==============================] - 0s 3ms/step - loss: 0.1394 - sparse_categorical_accuracy: 0.9772
Epoch 6/20
157/157 [==============================] - 0s 3ms/step - loss: 0.1900 - sparse_categorical_accuracy: 0.9776
Epoch 7/20
157/157 [==============================] - 0s 3ms/step - loss: 0.1077 - sparse_categorical_accuracy: 0.9834
Epoch 8/20
157/157 [==============================] - 0s 3ms/step - loss: 0.2491 - sparse_categorical_accuracy: 0.9668
Epoch 9/20
157/157 [==============================] - 0s 3ms/step - loss: 0.2112 - sparse_categorical_accuracy: 0.9736
Epoch 10/20
157/157 [==============================] - 0s 3ms/step - loss: 0.1949 - sparse_categorical_accuracy: 0.9724
Epoch 11/20
157/157 [==============================] - 0s 3ms/step - loss: 0.1078 - sparse_categorical_accuracy: 0.9806
Epoch 12/20
157/157 [==============================] - 0s 3ms/step - loss: 0.1012 - sparse_categorical_accuracy: 0.9850
Epoch 13/20
157/157 [==============================] - 0s 3ms/step - loss: 0.2211 - sparse_categorical_accuracy: 0.9714
Epoch 14/20
157/157 [==============================] - 0s 3ms/step - loss: 0.1620 - sparse_categorical_accuracy: 0.9768
Epoch 15/20
157/157 [==============================] - 0s 3ms/step - loss: 0.0882 - sparse_categorical_accuracy: 0.9834
Epoch 16/20
157/157 [==============================] - 0s 3ms/step - loss: 0.0205 - sparse_categorical_accuracy: 0.9944
Epoch 17/20
157/157 [==============================] - 0s 3ms/step - loss: 0.0522 - sparse_categorical_accuracy: 0.9942
Epoch 18/20
157/157 [==============================] - 0s 3ms/step - loss: 0.0528 - sparse_categorical_accuracy: 0.9916
Epoch 19/20
157/157 [==============================] - 0s 3ms/step - loss: 0.0507 - sparse_categorical_accuracy: 0.9908
Epoch 20/20
157/157 [==============================] - 0s 3ms/step - loss: 0.1889 - sparse_categorical_accuracy: 0.9756
Training done and variables updated
Trying to send newly updated model weights!
Successfully sent the weights
Training done, exiting!
PS C:\Users\Shubham\Desktop\Federated_Learning_using_mnist_data-main> 
```

Fig. 4. Client1 takes data range from 1 -2

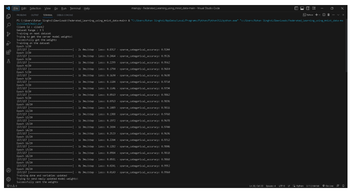
B. client 2

C. client 3

D. server

X. CONCLUSION

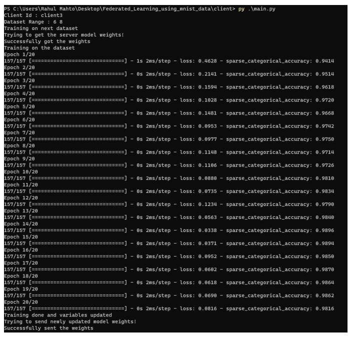
The project is a valuable contribution to the field of machine learning as it showcases the practical implementation of Federated Learning, a technique that has the potential to revolutionize the way machine learning models are trained on



File Edit Selection View Go Run Terminal Help
main.py - Federated_Learning_using_mnist_data-main - Visual Studio Code
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\Rohan Singhvi\Downloads\Federated_Learning_using_mnist_data-main> & "C:/Users/Rohan Singhvi/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Rohan Singhvi/Downloads/Federated_Learning_using_mnist_data-ma
in/client/main.py"
Client Id : client2
Dataset Range : 3 5
Training on next dataset
Trying to get the server model weights!
Successfully got the weights
Training on the dataset
Epoch 1/20
157/157 [==============================] - 2s 4ms/step - loss: 0.8317 - sparse_categorical_accuracy: 0.9244
Epoch 2/20
157/157 [==============================] - 1s 3ms/step - loss: 0.2464 - sparse_categorical_accuracy: 0.9526
Epoch 3/20
157/157 [==============================] - 1s 3ms/step - loss: 0.2299 - sparse_categorical_accuracy: 0.9562
Epoch 4/20
157/157 [==============================] - 1s 3ms/step - loss: 0.1790 - sparse_categorical_accuracy: 0.9650
Epoch 5/20
157/157 [==============================] - 1s 3ms/step - loss: 0.1630 - sparse_categorical_accuracy: 0.9630
Epoch 6/20
157/157 [==============================] - 1s 3ms/step - loss: 0.1104 - sparse_categorical_accuracy: 0.9714
Epoch 7/20
157/157 [==============================] - 1s 3ms/step - loss: 0.1146 - sparse_categorical_accuracy: 0.9734
Epoch 8/20
157/157 [==============================] - 1s 3ms/step - loss: 0.0563 - sparse_categorical_accuracy: 0.9862
Epoch 9/20
157/157 [==============================] - 1s 3ms/step - loss: 0.0763 - sparse_categorical_accuracy: 0.9836
Epoch 10/20
157/157 [==============================] - 1s 4ms/step - loss: 0.1009 - sparse_categorical_accuracy: 0.9816
Epoch 11/20
157/157 [==============================] - 1s 3ms/step - loss: 0.1388 - sparse_categorical_accuracy: 0.9768
Epoch 12/20
157/157 [==============================] - 1s 3ms/step - loss: 0.1972 - sparse_categorical_accuracy: 0.9678
Epoch 13/20
157/157 [==============================] - 1s 3ms/step - loss: 0.2036 - sparse_categorical_accuracy: 0.9700
Epoch 14/20
157/157 [==============================] - 1s 3ms/step - loss: 0.2523 - sparse_categorical_accuracy: 0.9636
Epoch 15/20
157/157 [==============================] - 1s 3ms/step - loss: 0.2206 - sparse_categorical_accuracy: 0.9712
Epoch 16/20
157/157 [==============================] - 1s 3ms/step - loss: 0.1202 - sparse_categorical_accuracy: 0.9806
Epoch 17/20
157/157 [==============================] - 1s 3ms/step - loss: 0.0984 - sparse_categorical_accuracy: 0.9818
Epoch 18/20
157/157 [==============================] - 0s 3ms/step - loss: 0.0581 - sparse_categorical_accuracy: 0.9868
Epoch 19/20
157/157 [==============================] - 0s 3ms/step - loss: 0.0241 - sparse_categorical_accuracy: 0.9952
Epoch 20/20
157/157 [==============================] - 0s 3ms/step - loss: 0.0143 - sparse_categorical_accuracy: 0.9960
Training done and variables updated
Trying to send newly updated model weights!
Successfully sent the weights
```

Fig. 5. Client2 takes data range from 3 - 5



```
PS C:\Users\Rahul Mahto\Desktop\Federated_Learning_using_mnist_data\client> py .\main.py
Client Id : client3
Dataset Range : 6 8
Training on next dataset
Trying to get the server model weights!
Successfully got the weights
Training on the dataset
Epoch 1/20
157/157 [==============================] - 1s 2ms/step - loss: 0.4628 - sparse_categorical_accuracy: 0.9414
Epoch 2/20
157/157 [==============================] - 0s 2ms/step - loss: 0.2141 - sparse_categorical_accuracy: 0.9514
Epoch 3/20
157/157 [==============================] - 0s 2ms/step - loss: 0.1594 - sparse_categorical_accuracy: 0.9618
Epoch 4/20
157/157 [==============================] - 0s 2ms/step - loss: 0.1028 - sparse_categorical_accuracy: 0.9720
Epoch 5/20
157/157 [==============================] - 0s 2ms/step - loss: 0.1481 - sparse_categorical_accuracy: 0.9668
Epoch 6/20
157/157 [==============================] - 0s 2ms/step - loss: 0.0953 - sparse_categorical_accuracy: 0.9742
Epoch 7/20
157/157 [==============================] - 0s 2ms/step - loss: 0.0977 - sparse_categorical_accuracy: 0.9750
Epoch 8/20
157/157 [==============================] - 0s 2ms/step - loss: 0.1148 - sparse_categorical_accuracy: 0.9714
Epoch 9/20
157/157 [==============================] - 0s 2ms/step - loss: 0.1106 - sparse_categorical_accuracy: 0.9726
Epoch 10/20
157/157 [==============================] - 0s 2ms/step - loss: 0.0880 - sparse_categorical_accuracy: 0.9810
Epoch 11/20
157/157 [==============================] - 0s 2ms/step - loss: 0.0735 - sparse_categorical_accuracy: 0.9834
Epoch 12/20
157/157 [==============================] - 0s 2ms/step - loss: 0.1234 - sparse_categorical_accuracy: 0.9790
Epoch 13/20
157/157 [==============================] - 0s 2ms/step - loss: 0.0563 - sparse_categorical_accuracy: 0.9840
Epoch 14/20
157/157 [==============================] - 0s 2ms/step - loss: 0.0338 - sparse_categorical_accuracy: 0.9896
Epoch 15/20
157/157 [==============================] - 0s 2ms/step - loss: 0.0371 - sparse_categorical_accuracy: 0.9894
Epoch 16/20
157/157 [==============================] - 0s 2ms/step - loss: 0.0952 - sparse_categorical_accuracy: 0.9850
Epoch 17/20
157/157 [==============================] - 0s 2ms/step - loss: 0.0602 - sparse_categorical_accuracy: 0.9870
Epoch 18/20
157/157 [==============================] - 0s 2ms/step - loss: 0.0618 - sparse_categorical_accuracy: 0.9864
Epoch 19/20
157/157 [==============================] - 0s 2ms/step - loss: 0.0690 - sparse_categorical_accuracy: 0.9862
Epoch 20/20
157/157 [==============================] - 0s 2ms/step - loss: 0.0816 - sparse_categorical_accuracy: 0.9816
Training done and variables updated
Trying to send newly updated model weights!
Successfully sent the weights
```

Fig. 6. Client takes data range from 6 - 8

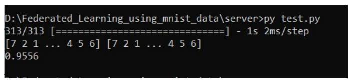
sensitive data while preserving privacy.
The project implements Federated Learning using the popular MNIST dataset, a standard benchmark dataset for handwritten digit recognition. The dataset consists of 60,000 training images and 10,000 test images, with each image being a 28x28 grayscale image of a handwritten digit.
The project's architecture is divided into two main parts: the client-side and the server-side. The client-side is responsible for training the model locally on the data available on the device, while the server-side aggregates the updates received from the clients and updates the global model.
The project's results demonstrate that Federated Learning is an effective technique for training machine learning models on distributed data while ensuring privacy. The project achieves an accuracy of 97.88 percent on the MNIST dataset, which is comparable to the accuracy obtained using traditional centralized learning methods.
One of the project's main advantages is that it demonstrates the potential of Federated Learning for real-world applications. In particular, Federated Learning can be used to train machine learning models on sensitive data such as medical records, financial data, and other forms of personally identifiable information, while preserving privacy and data ownership.

XI. Acknowledgment

I would like to express my sincere gratitude to all those who have contributed to the successful completion of this project.

First and foremost, I would like to thank Prof. KP Singh and Teaching Assistant Indrajeet sir and Avadhoot sir a for their guidance, support, and valuable feedback throughout the project. Their expertise and insights have been instrumental in shaping the direction of this research.

I would also like to thank my colleagues and classmates for their invaluable assistance and encouragement. Their willingness to share ideas and knowledge has been an essential factor in the success of this project.



```
D:\Federated_Learning_using_mnist_data\server>py test.py
313/313 [==============================] - 1s 2ms/step
[7 2 1 ... 4 5 6] [7 2 1 ... 4 5 6]
0.9556
```

Fig. 7. Server Predicts results with 95.56 %

References

[1] Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., Bacon, D. (2016, October). Federated learning: Strategies for improving communication efficiency. In NIPS workshop on private multi-party machine learning (pp. 1-8).
[2] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., Arcas, B. A. (2017, April). Communication-efficient learning of deep networks from decentralized data. In Artificial Intelligence and Statistics (pp. 1273-1282).
[3] Yang, Q., Liu, Y., Chen, T., Tong, Y. (2019). Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology (TIST), 10(2), 1-19.
[4] MNIST dataset: http://yann.lecun.com/exdb/mnist/
[5] Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., ... Lloyd, B. (2019). Practical secure aggregation for privacy-preserving machine learning. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (pp. 1175-1191).