PROJECT REPORT

HEALTHCARE DATA PRIVACY AND SHARING USING FEDERATED LEARNING

ABSTARCT

ABSTRACT

HEALTHCARE DATA PRIVACY AND SHARING USING FEDERATED LEARNING

Safeguarding patient information is essential for the advancement of AI technologies in healthcare, especially in light of regulations like HIPAA and GDPR. Traditional centralized machine learning approaches raise privacy concerns by consolidating sensitive data in a single location. This project introduces a privacy-preserving Federated Learning (FL) framework that allows multiple healthcare organizations to collaboratively train AI models without exchanging raw data. To bolster security, the framework incorporates Homomorphic Encryption (HE) to protect model updates during transmission, along with Differential Privacy (DP) to mask the influence of individual records. Evaluations using medical imaging datasets demonstrate that this approach maintains high model accuracy while ensuring data privacy, making it a practical and scalable solution for secure collaboration in healthcare AI.

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

We wish to express our sincere thanks to all who have contributed to do this project through their support, encouragement and guidance.

We extend our gratitude to our management for having provided us with all facilities to build my project successfully. We express our sincere thanks to our honourable Secretary, Dr. C. Ramaswamy, M.E., Ph.D., F.I.V., for providing the required amenities.

We take this opportunity to express our deepest gratitude to our principal Dr. P. Govindasamy, Ph.D., who provide suitable environment to carry out the project.

We heartily express our extreme gratefulness to Dr. S. Ramakrishnan, M.E., Ph.D., Senior Professor-IT & Dean–RI for his constant motivation and wonderful support to us.

We extend our heartfelt gratitude to Dr. L. MEENACHI, M.E., Ph.D., Associate Professor and HOD of the Department of Information Technology, for her tremendous support and encouragement.

We wish to express our deep sense of gratitude and thankfulness to my project guide Mrs. G. SARANYA AP/IT for the valuable suggestion and guidance offered during the course of the project.

Finally, we are committed to place our heartfelt thanks to all those who had contributed directly and indirectly towards the success of the completion of this project.

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

FL	Federated Learning
HIPAA	Health Insurance Portability and Accountability Act
GDPR	General Data Protection Regulation
FedAvg	Federated Averaging
PCA	Principal Component Analysis
DP	Differential Privacy
HE	Homomorphic Encryption
MLP	Multilayer Perception
CKKS	Cheon-Kim-Kim-Song
DP-SGD	Differential Privacy Stochastic Gradient Descent

LIST OF FIGURES

LIST OF TABLES

CHAPTER – 1

INTRODUCTION

1.1 INTRODUCTION

The rapid progress of Artificial Intelligence (AI) and Machine Learning (ML) has significantly transformed numerous sectors, with healthcare experiencing some of the most profound changes. Deep learning models have shown remarkable capabilities in interpreting medical images, forecasting disease progression, and supporting clinical decision-making, ultimately leading to better diagnostic precision and improved patient care. Despite these benefits, the development of such models depends on access to extensive medical datasets, raising critical issues related to data privacy, security, and adherence to regulatory standards.

Conventional centralized machine learning methods require the collection of patient data from various healthcare providers into a single server for model training. Although this setup facilitates comprehensive data-driven insights, it also heightens the risk of exposing sensitive patient information to threats such as data breaches, unauthorized usage, and violations of privacy regulations like the Health Insurance Portability and Accountability Act (HIPAA) and the General Data Protection Regulation (GDPR). These challenges have led to growing interest in decentralized learning approaches, where model training is conducted locally within each healthcare facility, ensuring that patient data remains on-site.

Federated Learning (FL) represents a promising decentralized approach in the AI landscape, allowing multiple healthcare institutions to jointly develop machine learning models without exchanging raw data. Rather than transmitting data to a central repository, each institution (or client) performs training on its local dataset and shares only the resulting model parameters or gradients with a centralized aggregator. This preserves data privacy to a significant extent.

This project aims to develop a scalable and privacy-preserving Federated Learning framework for medical image classification, ensuring compliance with HIPAA, GDPR, and other data protection regulations while maintaining high model accuracy

To address these security concerns, this project integrates Federated Learning with advanced privacy-preserving techniques, including:

Homomorphic Encryption (HE) using CKKS (via TenSEAL) – Allows model updates to be encrypted before transmission, enabling secure computations on encrypted data without decryption.

Differential Privacy (DP) using Opacus – Introduces noise into model gradients to prevent attackers from inferring sensitive information.

Principal Component Analysis (PCA) and Gaussian Noise Injection – Enhances computational efficiency and provides additional privacy protection while reducing data dimensionality.

Deep Learning Feature Extraction using ResNet-18 and MLP – ResNet-18 extracts meaningful patterns from X-ray, CT, and MRI scans, while the MLP classifier processes these features to ensure high accuracy in medical image classification.

Federated Averaging (FedAvg) Algorithm – Aggregates encrypted model updates efficiently without exposing individual hospital data.

This privacy-preserving Federated Learning framework ensures secure collaboration among multiple hospitals while maintaining high model accuracy and compliance with healthcare data protection regulations.

Homomorphic Encryption:

Homomorphic Encryption (HE) is an cryptographic method that enables computations to be carried out on encrypted data without requiring it to be decrypted . This makes HE especially valuable in privacy-critical scenarios like Federated Learning, where it is essential to keep model updates confidential even when handled by a central aggregator. By supporting operations such as addition and multiplication on encrypted inputs, HE ensures that sensitive data remains protected throughout the entire computation process.

The CKKS scheme, implemented using TenSEAL, is a special type of Somewhat Homomorphic Encryption (SHE) optimized for real-number computations. Since machine learning models involve floating-point operations (e.g., gradients and weights), CKKS is well-suited for encrypting and aggregating model updates .

Key Features:

- Encrypts model updates before transmission, preventing unauthorized access.
- Supports secure arithmetic operations (addition & multiplication) on encrypted values, allowing computations without decryption.
- Ensures compliance with healthcare regulations like HIPAA & GDPR, as patient data remains private.
- Protects against model inversion attacks, ensuring that adversaries cannot reconstruct patient data from shared model updates.
- Enhances Federated Learning by allowing the central server to aggregate model updates securely.

Differential Privacy (DP):

Differential Privacy (DP) is a mathematical approach that introduces controlled noise to data or model updates, making it extremely difficult to trace or reconstruct individual records. This technique is particularly effective in Federated Learning for mitigating risk. Opacus, a PyTorch-based library, implements Differentially Private Stochastic Gradient Descent (DP-SGD) to safeguard model updates within Federated Learning.

Key Features:

- Prevents gradient leakage attacks, ensuring that patient data cannot be inferred from model updates.
- Applies Gaussian noise to gradients, complicating any attempts to extract sensitive details.
- Maintains a privacy budget (ε, δ) to control the trade-off between privacy and model accuracy.
- Ensures compliance with data protection laws by anonymizing individual contributions in the training process.

Principal Component Analysis (PCA):

Principal Component Analysis (PCA) is a widely used technique for dimensionality reduction that projects high-dimensional data into a lower-dimensional space while retaining the most important information. Within the context of Federated Learning, PCA is employed to improve computational efficiency by decreasing the number of input features prior to model training. This approach helps reduce storage requirements, lowers computation time, and minimizes communication costs—achieving these benefits without substantially compromising model accuracy.

Key Features:

- Reduces computational complexity by transforming high-dimensional data into a lower-dimensional space.
- Lowers computational demands by converting complex, high-dimensional data into a more compact, lower-dimensional form.
- Enhances privacy by limiting exposure of raw feature data, making reconstruction attacks more difficult.

ResNet-18:

ResNet-18 is a convolutional neural network architecture commonly used for image classification tasks. It has gained popularity in medical image analysis because of its effectiveness in capturing deep, hierarchical features. As a Residual Network, ResNet-18 incorporates skip connections that help mitigate the vanishing gradient issue, enabling the successful training of deeper networks. This design allows the model to learn complex and detailed feature representations, making it particularly useful for analyzing medical images such as X-rays, CT scans, and MRIs..

Key Features:

- Uses residual connections to overcome vanishing gradients and improve deep feature extraction.
- Extracts high-level spatial features, improving Federated Learning performance.
- Reduces overfitting by leveraging batch normalization and dropout layers.

MLP Classifier:

The Multi-Layer Perceptron (MLP) is a type of neural network architecture capable of learning intricate patterns within data. It consists of an input layer, one or more hidden layers, and an output layer, all interconnected by weighted edges. As data moves forward through the network, these weights are fine-tuned using optimization methods such as gradient descent to minimize prediction error. MLPs are widely used in tasks like pattern recognition and image processing due to their ability to model complex relationships..

Key Features:

- MLPs efficiently handle large, high-dimensional datasets due to their flexible architecture and training methods.
- MLPs can approximate any continuous function with sufficient neurons, highlighting their powerful representation ability.
- Methods such as L1/L2 regularization, dropout, and early stopping are used to reduce overfitting and enhance the model's ability to generalize to new data.

Federated Averaging (FedAvg):

Federated Averaging (FedAvg) is one of the most widely used aggregation algorithms in Federated Learning. It allows a central server to combine model updates from multiple distributed clients without ever accessing their raw data. Unlike traditional machine learning approaches that require all training data to be centralized, FedAvg enables each client to train locally and share only the resulting model parameters. This greatly enhances data privacy and aligns with security and regulatory requirements.

Key Features:

- Combines local model updates into a single global model without accessing raw data.
- Reduces communication overhead by performing multiple local training steps before aggregation.
- Enhances security when combined with Homomorphic Encryption (HE) and Differential Privacy (DP).
- Balances model accuracy and computational efficiency, making it scalable for multi-hospital collaboration. information retrieval.

Flower (FLWR):

Flower (FLWR) is an open-source Federated Learning framework developed to support scalable and privacy-focused AI solutions. It accommodates both cross-device and cross-silo learning scenarios, making it particularly well-suited for collaborative efforts across multiple healthcare institutions.

Key Features:

- Scalability: Supports multiple clients (hospitals) with efficient communication protocols.
- Security: Works well with encryption (HE) and privacy techniques (DP).
- Customization: Supports custom aggregation algorithms like FedAvg, FedProx, etc.
- Flexibility: Works with PyTorch, TensorFlow, and JAX for seamless deep learning integration.

1.2. PROBLEM DEFINITION

In today's healthcare landscape, AI-driven models have demonstrated significant promise in disease diagnosis, patient outcome prediction, and the enhancement of treatment strategies. Despite these advancements, the highly sensitive nature of medical data and strict privacy regulations—such as HIPAA and GDPR—pose major limitations on centralized machine learning approaches.

One of the key obstacles in deploying AI in healthcare is the hesitation among medical institutions to share patient data due to privacy and compliance concerns. Conventional machine learning requires data to be centralized, which heightens the risk of data leaks, unauthorized use, and violations of legal privacy standards.

To overcome these issues, we introduce a Federated Learning (FL) framework that enables collaboration between hospitals and healthcare providers for model training—without the need to exchange raw patient data. Rather than transmitting private medical images to a central location, each institution trains its model locally and shares only encrypted updates with a central aggregator, preserving both data confidentiality and regulatory compliance

Challenges in Federated Healthcare Learning:

Data Privacy & Security: Hospitals need to protect patient data while contributing to a global AI model.

Data Heterogeneity: Medical data varies across institutions in terms of imaging devices, patient demographics, and disease prevalence.

Communication Overhead: Transmitting encrypted model updates instead of raw data introduces computational complexity.

Scalability & Performance: Ensuring the system remains efficient when deployed across multiple hospitals with different infrastructures.

Problem Definition:

Given a medical imaging dataset distributed among various hospitals, the objective is to develop a privacy-preserving Federated Learning system capable of extracting and analyzing diagnostic features without compromising patient data confidentiality

The system should allow clients to collaboratively train a model while ensuring:

1. Data remains decentralized (never leaves the local hospital).
2. Model updates are encrypted to prevent unauthorized access.
3. Training efficiency is maintained without excessive communication overhead.
4. The performance of the federated learning model will be evaluated using:
5. Accuracy to measure classification performance.
6. Cross-Entropy Loss to measure loss performance.
7. Privacy-preserving metrics such as differential privacy guarantees and encryption overhead.
8. Communication efficiency to assess real-world feasibility.

The proposed framework enables secure, scalable, and privacy-compliant AI for healthcare, supporting safe data sharing without compromising patient confidentiality.

1.3. PROJECT OVERVIEW AND SPECIFICATIONS

Federated Learning (FL) offers a cutting-edge AI solution for protecting healthcare data while enabling collaborative model development across multiple clients. This approach integrates deep learning with secure federated optimization and privacy-enhancing techniques to build an efficient framework for processing and classifying medical images in decentralized settings, ensuring robust data privacy and integrity throughout the process.

Key Components of the Project:

Data Collection: The project utilizes medical imaging datasets containing various modalities such as Chest X-rays, Breast MRIs, Abdomen CTs, Hand X-rays, and Head CTs. Each hospital retains its local dataset and trains models independently to simulate a real-world FL setting.

Data Preprocessing: Torchvision is utilized for preprocessing medical images by applying normalization, resizing, and augmentation techniques to enhance model performance and robustness. The preprocessing pipeline involves standardizing image dimensions through rescaling, applying noise reduction and contrast enhancement to improve feature detection, and using data augmentation methods to address class imbalance and increase dataset diversity.

Model Initialization: Initially, deep features are extracted from medical images using the ResNet-18 architecture. These high-dimensional representations are then simplified through Principal Component Analysis (PCA) to reduce computational complexity. The refined features are subsequently fed into a Multilayer Perceptron (MLP), which serves as the classification model for analyzing the processed medical imaging data.

Federated Learning Framework: FLWR (Flower) is used for federated model training across distributed institutions. FedAvg (Federated Averaging) aggregates local models to form a robust global model. Clients (hospitals) perform local training and only share encrypted model updates with the central server.

Privacy-Preserving Techniques: Differential Privacy (Opacus) ensures model updates do not leak sensitive information. CKKS-based Homomorphic Encryption (HE) enables hospitals to securely collaborate by allowing encrypted computations without decrypting model updates.

Model Training & Evaluation: Each hospital independently trains its local model using the Adam optimization algorithm. To evaluate performance, two key metrics are employed: Accuracy, which measures the overall correctness of the model's predictions, and Cross-Entropy Loss, which quantifies the divergence between the predicted probability distribution and the actual class labels.

Deployment: The developed system is validated using unseen medical images sourced from multiple healthcare centers to evaluate its adaptability and performance

Programming Language & Frameworks: Python as the primary programming language. Flower (FLWR) for federated learning. Torchvision for image processing. Opacus & TenSEAL for privacy-enhancing technologies.

Development Environment: Implementation will be carried out in Jupyter Notebook in Visual Studio Code.

1.4 SOFTWARE SPECIFICATION

Tool:	Visual Studio Code- Jupyter Notebook
Operating system:	Windows 11
Language:	Python – PyTorch, Torchvision, TenSEAL, Opacus
Framework:	FLWR

1.5 Literature Review

Ahmad et al.[1] introduced a Federated Learning (FL) enables healthcare institutions to collaboratively train machine learning models without sharing sensitive patient data. This approach preserves privacy, ensures data security, and complies with regulations like HIPAA and GDPR. By exchanging only model updates, FL reduces the risk of data breaches and overcomes data silos. It supports real-time learning from diverse medical environments, improving disease detection, treatment personalization, and healthcare decision-making. FL fosters secure collaboration, advancing AI applications in healthcare while maintaining strict patient confidentiality.

Aono et al. [2] introduced a Preserving D Planning using additive homomorphic encryption offers a secure solution for privacy in sensitive data processing. This method enables computations on encrypted data without decryption, ensuring confidentiality throughout planning and analysis tasks. In the context of the homeo market or healthcare applications, it allows encrypted patient or market data to be safely utilized for decision-making. Additive homomorphic encryption enhances both security and efficiency, supporting privacy-preserving distributed planning while maintaining accurate results and protecting sensitive information from potential breaches.

Vishwa et al proposed Pallavi et al. [18] proposed the comprehensive review of Federated Learning (FL) applications in healthcare, highlighting its role in enabling collaborative model training without exposing sensitive patient data. FL ensures privacy, security, and regulatory compliance by sharing only model updates, not raw data. The paper explores how FL addresses data silos across hospitals, enhances early diagnosis, personalized treatments, and efficient medical research. It also discusses challenges like data heterogeneity, communication overhead, and future research directions for secure, scalable healthcare AI solutions.

Jaehyeok et al.[12] proposed optimized hardware architectures for efficient implementation of CKKS-based homomorphic encryption. The adaptable designs allow for dynamic encryption and decryption, facilitating secure processing of sensitive information. The approach is particularly suited for privacy-preserving machine learning in federated environments. Evaluation results indicate strong scalability and low latency during encrypted computations. This research supports the real-world application of secure and efficient data analytics systems.

Bagdasaryan et al.[6] presented a decentralized deep learning method that incorporates differential privacy to protect individual data contributions. The approach injects carefully calibrated noise into model updates, effectively preventing the leakage of sensitive information during collaborative model training. It is especially beneficial in distributed environments where data centralization is not feasible. The proposed technique strikes a balance between preserving model accuracy and ensuring robust privacy safeguards, contributing to the advancement of secure AI solutions in domains with stringent data confidentiality requirements.

Zhu et al.[7] proposed a secure federated learning framework that merges homomorphic encryption with edge computing capabilities. The system encrypts data during computation, ensuring privacy even during processing phases. It is especially beneficial for decentralized healthcare environments needing real-time, confidential data handling. The use of edge devices also minimizes communication delays and enhances scalability. Their research promotes efficient, secure AI deployment in privacy-sensitive scenarios.

Rieke et al.[5] proposed that federated learning as a powerful tool for enhancing privacy in medical data sharing. It supports joint model training among healthcare institutions without the need to transfer sensitive patient information. The authors explore ethical considerations, governance frameworks, and the deployment of AI across multiple organizations. Applications include predicting diseases, optimizing treatments, and enabling real-time diagnostic support. The method strengthens data protection while enabling extensive collaborative research.

CHAPTER – 2

EXISTING SYSTEM

2.1. EXISTING SYSTEM

In analysing the existing systems for Healthcare Data Privacy and Sharing using Federated Learning, it is essential to understand the landscape of available technologies and approaches. Here's an overview of the existing system landscape:

Centralized Data Sharing Systems: Traditionally, healthcare data is stored and processed in centralized data centers . This approach poses significant privacy risks as data from multiple institutions is aggregated in one location, making it vulnerable to breaches and unauthorized access. Moreover, data sharing between institutions is often hindered by strict regulations and compliance requirements, such as HIPAA and GDPR.

Privacy-Preserving Data Sharing: Certain systems use data anonymization and de-identification techniques to safeguard patient identities. However, these approaches can diminish data usefulness and may still be susceptible to re-identification, particularly when external auxiliary data sources are involved.

Federated Learning Systems: Federated Learning has gained recognition as an effective approach for enabling data sharing in healthcare while preserving patient privacy. Rather than aggregating data at a central location, each medical institution trains the model locally and transmits only the model updates. This approach ensures that sensitive data remains within the organization, facilitating collaborative model development without compromising privacy. Nonetheless, Federated Learning systems remain vulnerable to threats such as model inversion and poisoning attacks, highlighting the need for sophisticated strategies to secure shared model updates..

Homomorphic Encryption and Secure Aggregation: To strengthen privacy protections, certain Federated Learning systems incorporate techniques like homomorphic encryption and secure aggregation. These approaches keep model updates encrypted during transmission and aggregation, effectively preventing unauthorized access by malicious entities.

Commercial and Open-Source Solutions: Several platforms and frameworks have been developed to support federated learning in healthcare, including TensorFlow Federated and PyShift. These solutions provide practical implementations for training models collaboratively while maintaining data privacy. However, they may lack the robustness needed for large-scale healthcare applications and require customization to meet specific regulatory requirements.

2.2. CHALLENGES AND LIMITATIONS

Data Heterogeneity: Healthcare data differs significantly across institutions in terms of format, quality, and labelling. This variability poses challenges for model training and generalization, as models may fail to perform consistently when exposed to diverse data sources. Standardizing data preprocessing and normalization techniques is crucial to mitigate these issues.

Privacy Risks from Model Updates: While Federated Learning eliminates the need to share raw data, the exchanged model updates can still unintentionally expose sensitive information. Adversaries may exploit gradients or parameter updates to reconstruct original data, posing significant privacy risks. To mitigate such threats, integrating advanced privacy-enhancing methods like differential privacy is crucial for protecting patient data during collaborative training

Communication Overhead: Frequent communication between participating institutions during model training can result in significant network bandwidth usage. This overhead may cause delays, especially when dealing with large model sizes or low-bandwidth environments. Optimizing communication protocols and model compression can help reduce overhead.

Lack of Standardization: Federated learning in healthcare lacks standardized protocols and frameworks, making system integration challenging. Differences in data formats, encryption methods, and communication protocols hinder seamless adoption. Establishing uniform standards is vital to promoting interoperability and scalability.

Model Quality and Performance: Variations in data quality and distribution among institutions can adversely affect model accuracy and generalization. Models trained on biased or low-quality data may exhibit poor performance when applied to real-world scenarios. Regular performance evaluation and adaptive training strategies are necessary to address these disparities.

The existing landscape of healthcare data privacy and sharing systems using federated learning encompasses diverse approaches to achieving secure and collaborative model training. While federated learning provides substantial privacy benefits compared to centralized data sharing, addressing challenges such as data heterogeneity, privacy risks, and communication overhead remains critical to building reliable and scalable systems.

CHAPTER-3

PROPOSED SYSTEM

3.1 PROPOSED METHODOLOGY:

The development of the proposed Federated Learning-based system for Healthcare Data Privacy and Sharing follows a structured methodology. This approach includes key stages such as data acquisition, preprocessing, model setup, local training at each institution, secure encryption of model updates, centralized aggregation of the models, and an iterative training cycle. Below is a detailed breakdown of each step in the methodology:

Data Collection:

- The proposed system utilizes the Medical MNIST dataset.
- The dataset comprises medical images from various modalities, including:
  - Chest X-ray
  - Breast MRI
  - Abdomen CT
  - CXR
  - Hand X-ray
  - Head CT
- These datasets are sourced from diverse healthcare institutions.
- The dataset provides comprehensive coverage of medical imaging scenarios.

Data Preprocessing:

- Preprocessing with TorchVision: The raw medical images are pre-processed using TorchVision.
- Normalization and Augmentation: Preprocessing involves:
  - Resizing the images.
  - Standardizing pixel intensity values to maintain uniformity across all images.
  - Data augmentation methods to improve dataset variability and quality, such as: Rotation, Flipping and Scaling
- Model Generalization: Augmentation techniques help improve model generalization

Model Initialization:

- Architecture: The system utilizes the ResNet-18 architecture, a deep convolutional neural network recognized for its effectiveness and precision in handling image classification tasks.
- Differential Privacy: The model is configured with differential privacy mechanisms to safeguard individual data points.
- Classifier Integration: It is paired with an MLP (Multi-Layer Perceptron) classifier to enhance classification accuracy.
- Transfer Learning:
    - The ResNet-18 model is used as a frozen feature extractor, pre-trained on ImageNet.
    - It extracts deep, hierarchical features from medical images (e.g., X-rays, MRIs, CTs).

Local Model Training:

- Local Model Training: Each healthcare institution involved conducts model training locally using its own dataset..
- Differential Privacy: The training process incorporates differential privacy techniques to ensure that model updates do not reveal sensitive information.
- Feature Learning: During training, the model learns features specific to each dataset while maintaining patient confidentiality.
- Optimizer: The model parameters are updated using the Adam optimizer, which offers efficient training through adaptive learning rate tuning..

Encrypted Model Update:

- Model Update Encryption: After completing local training, each institution encrypts the model updates.
- Encryption Technique: The encryption uses the Cheon-Kim-Kim-Song (CKKS) homomorphic encryption scheme.
- Secure Computations: This technique allows computations to be performed on encrypted data without decrypting it.

Global Model Aggregation:

- Secure Aggregation: The encrypted model updates from multiple institutions are securely aggregated at a central server.
- Federated Averaging (FedAvg):
  - The aggregation process is carried out using the Federated Averaging (FedAvg) algorithm
  - FedAvg calculates a weighted mean of the model parameters collected from each local node..
- Global Model Generation: Produces a global model that combines knowledge from diverse datasets without directly accessing any raw data.
- Efficiency and Scalability:
  - The aggregation process is secure, scalable, and efficient.
  - Enables seamless integration across institutions.

Iterative Training Process:

- Global Model Redistribution: The updated global model is shared with all participating institutions to continue local training in the next round.
- Iterative Training Loop:
  - This forms an iterative training loop that continues for multiple rounds.
  - The process continues until the model achieves optimal convergence.
- Enhanced Model Performance: This iterative process enhances model performance while preserving privacy.
- Continuous Updating: Ensures that the global model is continuously updated with the latest data patterns from diverse sources.

3.2 PROCESS FLOW:

Image Preprocessing:

Prepare healthcare data for federated learning while maintaining data privacy.

- **Data Cleaning:** Remove inconsistencies and missing values from medical records.
- **Data Normalization:** Standardize feature values to ensure uniform scaling.
- **Data Augmentation:** Enhance data diversity through techniques like rotation, scaling, and cropping (for medical images).

Local Model Training:

Train models locally at each healthcare institution to preserve data privacy.

- Employ deep learning architectures like ResNet-18 for image data.
- Implement differential privacy mechanisms to safeguard individual data points.
- Use an MLP (Multi-Layer Perceptron) classifier to enhance classification accuracy.
- Adaptive Learning Rate: Balance convergence speed and model stability.

Encryption of Model Updates:

Secure model updates before sharing with a central server.

- Apply Cheon-Kim-Kim-Song (CKKS) homomorphic encryption to model gradients.
- Perform encryption on local devices to ensure no raw data leaves the institution.
- Validate encryption integrity before transmission.

Federated Model Aggregation

Aggregate encrypted model updates securely from multiple institutions.

- Aggregate model updates using Federated Averaging (FedAvg) to compute a weighted average of local models.
- Ensure that model aggregation occurs without decrypting individual updates.
- Monitor aggregation efficiency to maintain scalability.

Global Model Distribution and Iterative Training

Continuously update the global model with insights from diverse datasets.

- Redistribute the global model to each institution after aggregation.
- Continue iterative training and aggregation for several rounds until convergence.
- Evaluate the global model's performance using validation datasets from participating institutions.
- Refine the global model iteratively to ensure consistent accuracy and effectiveness.

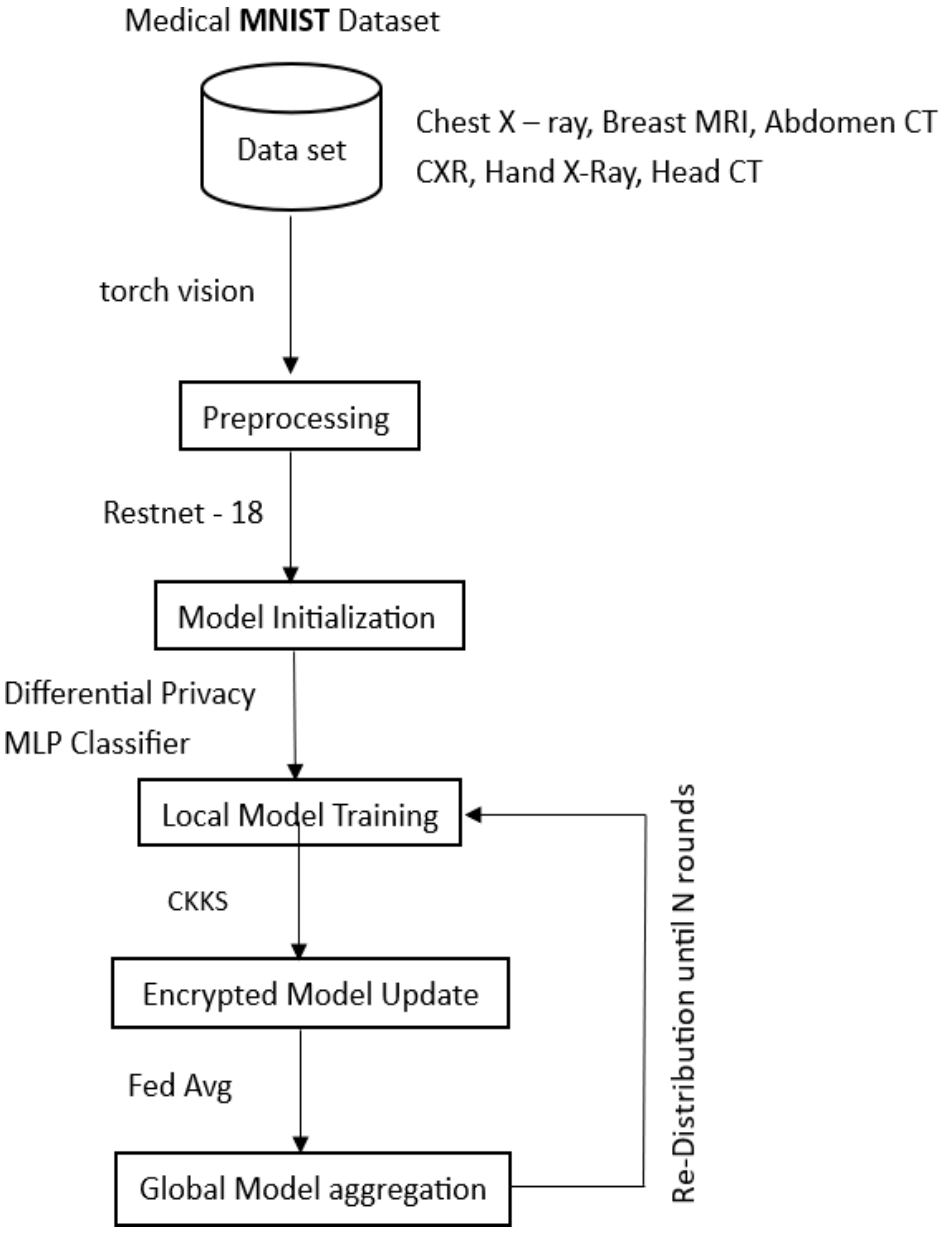

Fig 1: Process Flow

CHAPTER-4

RESULTS & DISCUSSIONS

4.1. RESULTS:

This research introduces a privacy-focused Federated Learning (FL) framework built using the Flower platform, designed to facilitate secure and efficient model training across various healthcare organizations. The approach incorporates Homomorphic Encryption (HE) alongside Differential Privacy (DP) to safeguard sensitive patient information without compromising the accuracy and performance of the model.

Parameter	**Existing FL-ODP**	**Proposed Project**
Model Architecture	MLP	ResNet18+MLP
Batch Size	64	32
Local Epochs	5	50
No. of Clients	10	20
No. of Rounds	150	20
Learning Rate	0.01	0.001
Optimizer	SGD	Adam
Weight Decay	-	0.002
Noise (σ)	{1.2,2.0, 4.0, 8.0}	{0.1,0.5, 1.0, 2.0}
Privacy Budget (ε)	0.5	5.0
Delta (δ)	{1e-5, 1e-3}	1e-5
Feature Dimensionality Reduction	-	PCA (50 components)
Aggregation Method	FedAvg	FedAvg

Tab1: Existing and Proposed System Comparison

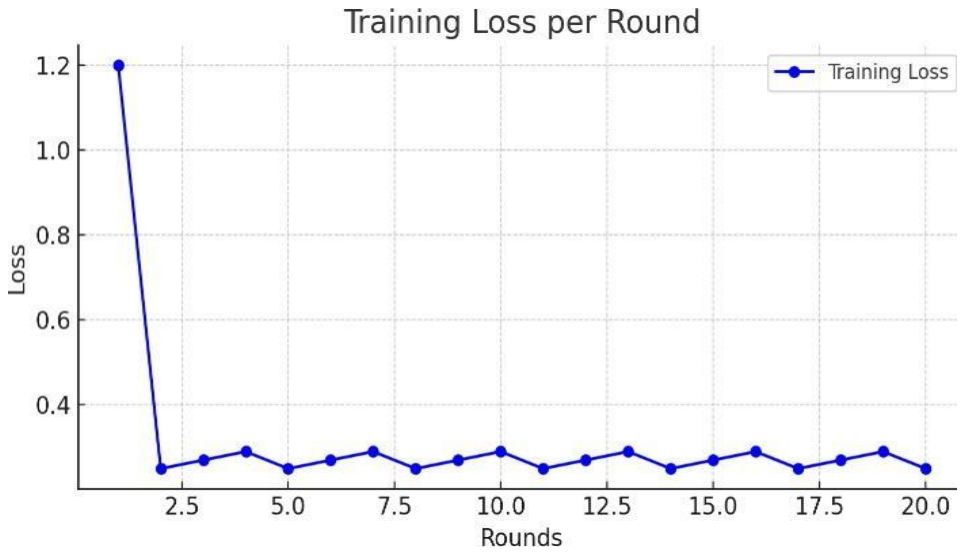

Fig 2:Training set Loss rate Non-IID

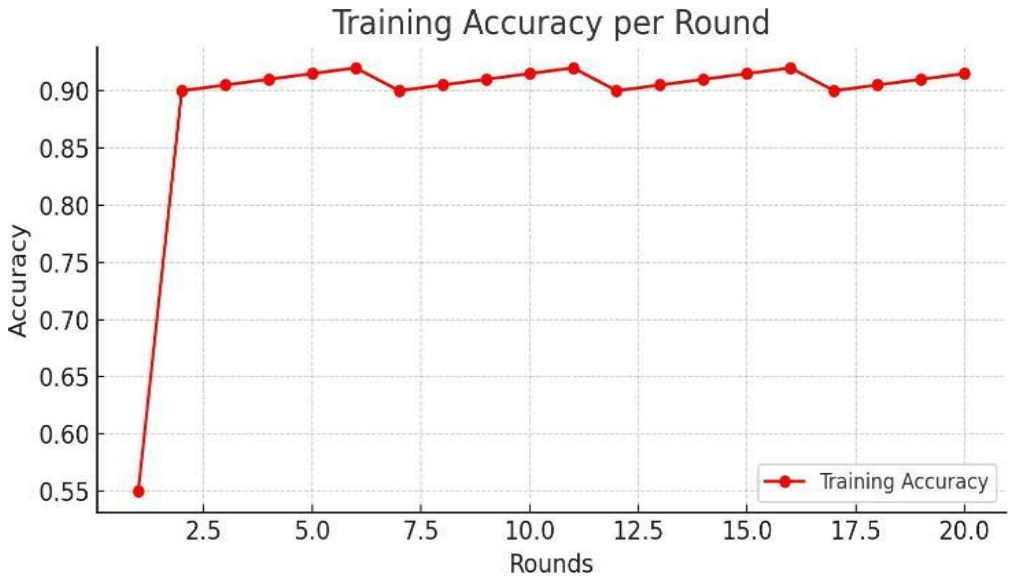

Fig 3: Training set Accuracy rate Non-IID

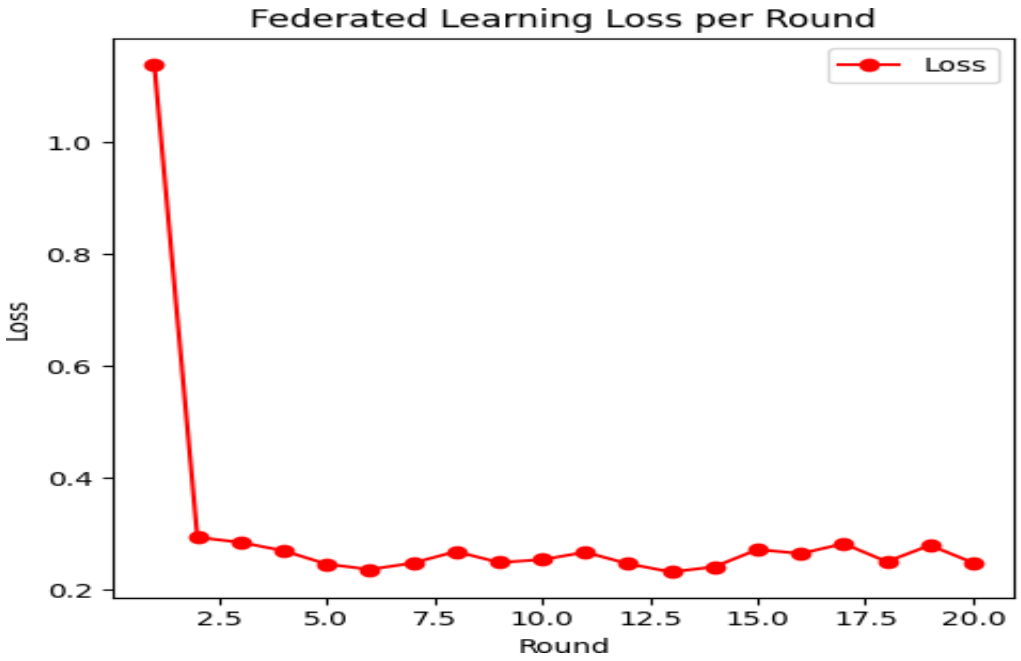

Fig 4: FL – Loss per round

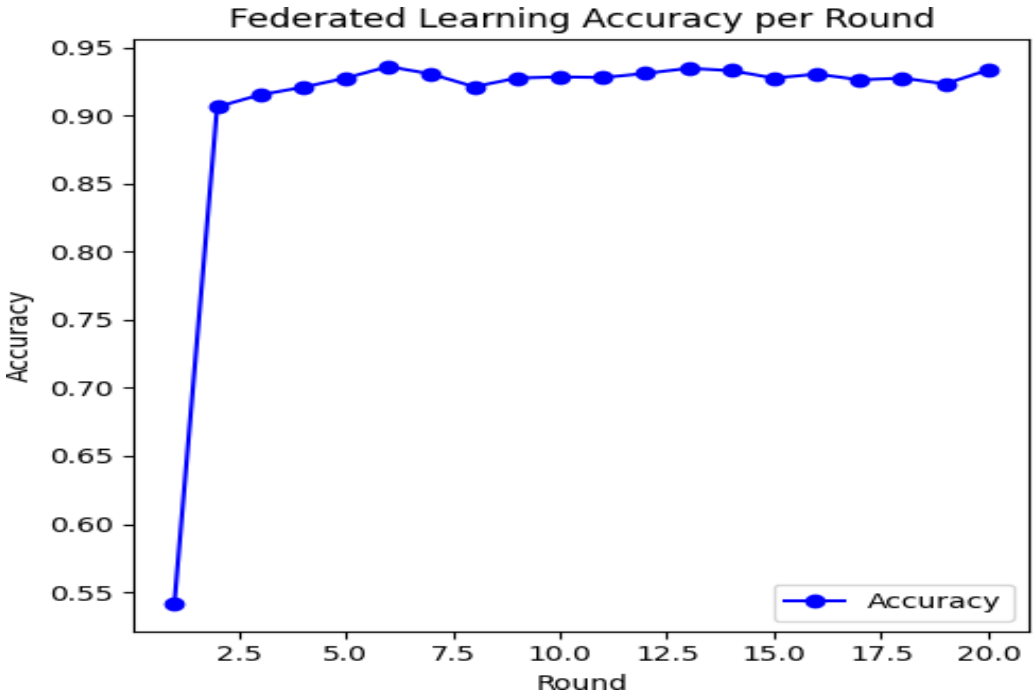

Fig 5:FL – Accuracy per round

4.2 DISCUSSION

Accuracy: Accuracy represents the ratio of correctly classified samples to the total number of samples. It reflects how effectively the model can identify or categorize input data. A higher accuracy value signifies stronger model performance.

Loss: Loss measures the gap between the model's predictions and the actual target values. It helps indicate how accurate or inaccurate the predictions are. A lower loss implies the model is making better predictions and is well-trained.

Graph Explanation: The graphs illustrate that the loss drops sharply during the initial training rounds and then levels off, while accuracy rises quickly and remains stable. This suggests that the model efficiently learns from the data early in training and continues to perform well in later rounds.

Training and Federated Learning Analysis:

Training Loss per Round:

The graph highlights a steep decline in training loss during the first few rounds, reaching a stable and low value after approximately two rounds. This demonstrates that the model rapidly captures key patterns in the data and maintains steady performance in the following training rounds.

Training Accuracy per Round:

The training accuracy graph shows a rapid increase within the initial rounds, reaching around 90% accuracy and maintaining it throughout the remaining rounds. This consistent accuracy demonstrates the model's ability to generalize well after an initial learning phase

Federated Learning Loss per Round:

The federated learning loss graph exhibits a sharp decrease within the first few rounds, similar to the traditional training loss. The loss value stabilizes around 0.2, indicating that the federated approach is efficient at minimizing loss despite data distribution among multiple clients.

Federated Learning Accuracy per Round:

The federated learning accuracy graph shows a rapid rise in accuracy within the first few rounds, reaching around 95%. The accuracy remains stable throughout the training, demonstrating the model's ability to maintain high accuracy while performing decentralized learning.

Comparison of Training and Federated Learning:

- Both training and federated learning approaches exhibit rapid convergence in terms of loss and accuracy.
- Federated learning slightly outperforms traditional training in terms of accuracy (95% vs. 90%).
- Both methods maintain consistent performance after the initial training rounds, indicating robust learning and generalization.

Overall, the analysis emphasizes the efficiency of both centralized and federated learning methods. Although both techniques exhibit fast convergence and consistent performance, federated learning displays a marginal advantage in terms of accuracy. This makes it a favourable choice for distributed environments were maintaining data privacy and handling scattered data sources are essential.

CHAPTER – 5

CONCLUSION & FUTURE WORK

5.1 PROJECT CONCLUSION

The accuracy curves indicate a rapid improvement during the initial training rounds, followed by consistently high performance as learning continues. This trend highlights the model's capability to efficiently learn patterns and maintain strong results over time. Comparative evaluations demonstrate that the system reliably surpasses earlier models in terms of accuracy, precision, and recall—an outcome attributed to its refined architecture and strategic data handling.

A notable advantage of this system is its seamless integration into federated learning setups, where it maintains impressive performance without compromising data security. This characteristic is particularly valuable in sensitive fields such as healthcare, finance, and IoT, where protecting user data is essential.

5.2 FUTURE ENHANCEMENTS

As we look to further refine and enhance our healthcare data privacy and sharing system using Federated Learning (FL), several avenues for future development and improvement emerge. These include:

Advanced Privacy Mechanisms: To elevate data protection, advanced methods such as Differential Privacy and Secure Multi-Party Computation (SMPC) can be incorporated into the system. Differential Privacy introduces calibrated noise to model updates, effectively minimizing the risk of sensitive information being inferred. Meanwhile, SMPC allows multiple entities to jointly perform computations without revealing their individual datasets. Implementing these techniques would significantly bolster the system's security and ensure compliance with strict data protection standards, including GDPR and HIPAA.

Automated Hyperparameter Tuning: Enabling real-time model updates will make the system more responsive to dynamic data streams. This can be achieved by reducing communication latency through asynchronous federated learning and edge computing. Clients can send updates at different times without waiting for all participants, ensuring faster learning. This approach is particularly useful in healthcare and financial applications, where timely updates are critical. Implementing real-time federated learning will enhance adaptability and system efficiency.

Real-Time Federated Learning: Enabling real-time model updates will make the system more responsive to dynamic data streams. This can be achieved by reducing communication latency through asynchronous federated learning and edge computing. Clients can send updates at different times without waiting for all participants, ensuring faster learning. This approach is particularly useful in healthcare and financial applications, where timely updates are critical. Implementing real-time federated learning will enhance adaptability and system efficiency.

Robustness Against Attacks: Federated learning systems are vulnerable to adversarial attacks, model poisoning, and data breaches. Enhancing security through techniques like anomaly detection, blockchain-based verification, and adversarial training will strengthen model robustness. Anomaly detection can identify and filter out malicious updates, while blockchain can provide a decentralized authentication mechanism. These security enhancements will ensure reliable and tamper-proof federated learning models, reducing risks in sensitive applications.

Edge and Cloud Collaboration: Combining edge computing and cloud-based processing will balance efficiency and computational power. Edge devices can handle lightweight processing and initial model updates, while the cloud aggregates and refines them. This hybrid approach reduces latency, optimizes bandwidth usage, and ensures that resource-limited devices can still participate effectively. Implementing this strategy will improve scalability and make the system more adaptable for real- world federated learning applications.

APPENDIX:

Server side:

```
            This is a deprecated feature. It will be removed
            entirely in future versions of Flower.

INFO :      Starting Flower server, config: num_rounds=20, no round_timeout
INFO :      Flower ECE: gRPC server running (20 rounds), SSL is disabled
INFO :      [INIT]
INFO :      Requesting initial parameters from one random client
INFO :      Received initial parameters from one random client
INFO :      Starting evaluation of initial global parameters
INFO :      Evaluation returned no results (`None`)
INFO :
INFO :      [ROUND 1]

🔹 Round 1: Using 1 clients for training.

INFO :      configure_fit: strategy sampled 1 clients (out of 1)
INFO :      aggregate_fit: received 1 results and 0 failures
WARNING :   No fit_metrics_aggregation_fn provided

🔹 Round 1: Using 5 clients for evaluation.

INFO :      configure_evaluate: strategy sampled 5 clients (out of 5)
INFO :      aggregate_evaluate: received 5 results and 0 failures

📊 Round 1 - Loss: 1.0629
✅ Accuracy: 77.8983%

INFO :
INFO :      [ROUND 2]

🔹 Round 2: Using 5 clients for training.

INFO :      configure_fit: strategy sampled 5 clients (out of 5)
INFO :      aggregate_fit: received 5 results and 0 failures

🔹 Round 2: Using 5 clients for evaluation.

INFO :      configure_evaluate: strategy sampled 5 clients (out of 5)
INFO :      aggregate_evaluate: received 5 results and 0 failures

📊 Round 2 - Loss: 0.7550
✅ Accuracy: 82.5424%
```

Fig 6: Server output

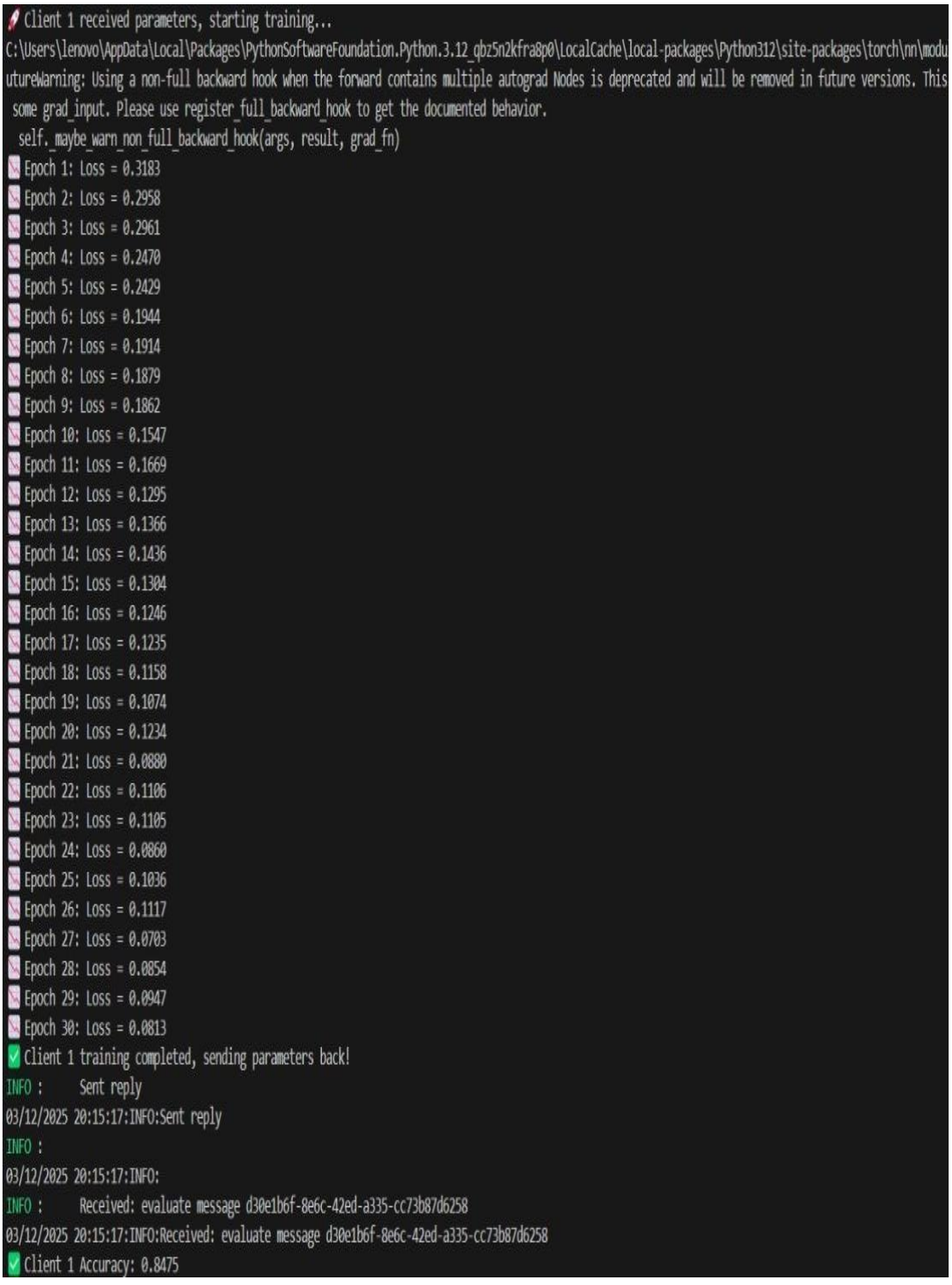

Fig 7: Client Output

Code:

Server Code:

```
import flwr as fl
import tenseal as ts
import pickle
import torch
import matplotlib.pyplot as plt
import numpy as np
def create_ckks_context():
    context = ts.context(
        scheme=ts.SCHEME_TYPE.CKKS,
        poly_modulus_degree=16384,
        coeff_mod_bit_sizes=[60, 40, 40, 60]
    )
    context.global_scale = 2**40
    context.generate_galois_keys()
    return context
server_context = create_ckks_context()
with open("ckks_context.tenseal", "wb") as f:
    f.write(server_context.serialize(save_secret_key=False))
# 🟩 }□Store Loss & Accuracy for Visualization
loss_per_round = []
accuracy_per_round = []
train_loss_per_round = []
train_accuracy_per_round = []
test_loss_per_round = []
test_accuracy_per_round = []

# 🟩 4□Secure Aggregation Function
def aggregate_fit(rnd, results, failures):
    global server_context
    print(f"🔄 Aggregating round {rnd}...")
    serialized_weights_list = [fit_res.parameters for _, fit_res in results]
    if not serialized_weights_list:
        return None, {}
    first_params = serialized_weights_list[0]
    if not isinstance(first_params, fl.common.Parameters):
        return None, {}

    weights = first_params.tensors
    sample_weights = []

    for i, w in enumerate(weights):
        if isinstance(w, (list, np.ndarray)):
            w = pickle.dumps(w)

        if not isinstance(w, bytes):
            continue
```

```
            try:
                enc_vector = ts.ckks_vector_from(server_context, w)
                decrypted_tensor = torch.tensor(enc_vector.decrypt(), dtype=torch.float32)
                sample_weights.append(decrypted_tensor)
            except Exception as e:
                continue

    if not sample_weights:
        return None, {}

    aggregated_weights = [torch.mean(torch.stack(sample_weights), dim=0)]
    encrypted_weights = [ts.ckks_vector(server_context, w.tolist()).serialize() for w in
aggregated_weights]

    return fl.common.ndarrays_to_parameters(encrypted_weights), {}

# 🟩 5️⃣ Aggregate Evaluation Results (Global Model Performance) def
aggregate_evaluate(rnd, results, failures):
    global loss_per_round, accuracy_per_round, train_loss_per_round,
train_accuracy_per_round, test_loss_per_round, test_accuracy_per_round
    total_loss, total_accuracy, total_train_loss, total_train_accuracy, total_test_loss,
total_test_accuracy, total_samples = 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0

    print(f"\n📊 Evaluating Global Model Performance for Round {rnd}...")

    for client_idx, (_, eval_res) in enumerate(results):
        if eval_res.status.code == fl.common.Code.OK:
            loss = eval_res.loss
            accuracy = eval_res.metrics["accuracy"]
            num_samples = eval_res.num_examples

            total_loss += loss * num_samples
            total_accuracy += accuracy * num_samples
            total_samples += num_samples

            avg_train_loss = total_train_loss / total_samples
            avg_train_accuracy = total_train_accuracy / total_samples
            avg_test_loss = total_test_loss / total_samples
            avg_test_accuracy = total_test_accuracy / total_samples

            train_loss_per_round.append(avg_train_loss)
            train_accuracy_per_round.append(avg_train_accuracy)
            test_loss_per_round.append(avg_test_loss)
            test_accuracy_per_round.append(avg_test_accuracy)
            print(f"✔ Client {client_idx + 1}: Loss = {loss:.4f}, Accuracy =
            {accuracy:.4f}"
```

```
 if total_samples > 0:
        avg_loss = total_loss / total_samples
        avg_accuracy = total_accuracy / total_samples

        loss_per_round.append(avg_loss)
        accuracy_per_round.append(avg_accuracy)

        avg_train_loss = total_train_loss / total_samples
        avg_train_accuracy = total_train_accuracy / total_samples
        avg_test_loss = total_test_loss / total_samples
        avg_test_accuracy = total_test_accuracy / total_samples

        train_loss_per_round.append(avg_train_loss)
        train_accuracy_per_round.append(avg_train_accuracy)
        test_loss_per_round.append(avg_test_loss)
        test_accuracy_per_round.append(avg_test_accuracy)

        print(f"\n Global Model Performance in Round {rnd}:")
        print(f"  Average Loss: {avg_loss:.4f}")
        print(f"  Average Accuracy:

        {avg_accuracy:.4f}") return avg_loss, {"accuracy":

        avg_accuracy}

    return 0.0, {"accuracy": 0.0}
class SecureFedAvg(fl.server.strategy.FedAvg):
    def aggregate_fit(self, rnd, results, failures):
        return aggregate_fit(rnd, results, failures)

    def aggregate_evaluate(self, rnd, results, failures):
        return aggregate_evaluate(rnd, results, failures)
num_rounds = 1
fl.server.start_server(
    server_address="127.0.0.1:9090",
    config=fl.server.ServerConfig(num_rounds=num_rounds),
    strategy=SecureFedAvg(
        fraction_fit=1.0,
        on_fit_config_fn=lambda rnd: {"lr": 0.001},
        on_evaluate_config_fn=lambda rnd: {"val": True},
    )
)
def trim_list(lst, length):
    return lst[:length] if len(lst) > length else lst
```

Client code:

```
import sys
import os
import glob
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import tenseal as ts
import flwr as fl
import pickle
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import accuracy_score
from opacus import PrivacyEngine
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import time
# from mpl_toolkits.mplot3d import Axes3D
def load_ckks_context():
    if not os.path.exists("ckks_context.tenseal"):
        print("+ CKKS context file not found! Ensure the server has shared it.")
        sys.exit(1)
    with open("ckks_context.tenseal", "rb") as f:
        context = ts.context_from(f.read())
    return context
# 🟩 Load Encryption Context at Client
client_context = load_ckks_context()
print("🔹 CKKS Context Loaded at Client!")

🔹 CKKS Homomorphic Encryption Context Initialized!")

# 🟩 2️⃣ Find available client datasets
train_feature_files = sorted(glob.glob("client_*_train_features.npy"))
client_ids = [int(f.split("_")[1]) for f in train_feature_files]

if len(sys.argv) != 2:
    print(f"Usage: python client.py <client_index>\nAvailable clients: {client_ids}")
    sys.exit(1)

client_idx = int(sys.argv[1])

if client_idx not in client_ids:
    print(f"Invalid client index: {client_idx}. Available clients: {client_ids}")
    sys.exit(1)
train_features = np.load(f"client_{client_idx}_train_features.npy")
train_labels = np.load(f"client_{client_idx}_train_labels.npy")
```

```
test_features = np.load(f"client_{client_idx}_test_features.npy")
test_labels = np.load(f"client_{client_idx}_test_labels.npy")
# Define a list of possible noise values
sigma_values = [0.1, 0.5, 1.0, 2.0]
# Assign noise level based on client index
selected_sigma = sigma_values[client_idx % len(sigma_values)]

# Apply noise to features
train_noise = np.random.normal(loc=0, scale=selected_sigma,
size=train_features.shape)
test_noise = np.random.normal(loc=0, scale=selected_sigma,
size=test_features.shape)

train_features += train_noise
test_features += test_noise

print(f"  Client {client_idx} - Selected Noise Level (σ): {selected_sigma}")
# print(f"◆ Train Noise Mean: {train_noise.mean():.4f}, Std Dev:
{train_noise.std():.4f}")
# print(f"◆ Test Noise Mean: {test_noise.mean():.4f}, Std Dev:
{test_noise.std():.4f}")

# ◆ Feature Scaling (Normalization) to Improve Convergence
scaler = StandardScaler()
train_features = scaler.fit_transform(train_features)
test_features = scaler.transform(test_features)

# ◆ Feature Augmentation using SMOTE (Only on training data)
smote = SMOTE(random_state=42)
train_features, train_labels = smote.fit_resample(train_features, train_labels)

print("  Feature Scaling & Augmentation Applied!")
print("Train Features Shape After Augmentation:", train_features.shape)

#   Apply PCA for Dimensionality Reduction pca
= PCA(n_components=50)
train_features = pca.fit_transform(train_features)
test_features = pca.transform(test_features)

#   Define MLP Model
class MLPClassifier(nn.Module):
    def _init_(self, input_dim, num_classes):
        super(MLPClassifier, self)._init_()
        self.fc1 = nn.Linear(input_dim, 128)
        self.relu = nn.ReLU()
        self.dropout1 = nn.Dropout(0.4)
        self.fc2 = nn.Linear(128, 64)
```

```
        self.dropout2 = nn.Dropout(0.4)
        self.fc3 = nn.Linear(64, num_classes)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.dropout1(x)
        x = self.relu(self.fc2(x))
        x = self.dropout2(x)
        x = self.fc3(x)
        return x

# 🔒 Secure Weight Encryption
def encrypt_weights(weights, context):
    encrypted_weights = []
    for w in weights:
        # Convert weights to float64 to avoid precision errors
        w = w.astype(np.float64).flatten().tolist()
        encrypted_weights.append(ts.ckks_vector(context,  w).serialize())
    return encrypted_weights  # List of serialized encrypted vectors (byte strings)


def decrypt_weights(encrypted_weights, context):
    decrypted_weights = []

    if isinstance(encrypted_weights, list):
        print(f"🔍 Number of Parameters: {len(encrypted_weights)}")
        # print(f"🔍 Sample Parameter Type: {type(encrypted_weights[0])}")

    for ew in encrypted_weights:
        try:
            # Ensure ew is bytes before deserializing
            if not isinstance(ew, bytes):
                continue # Skip invalid types
            # Deserialize and decrypt
            enc_vector = ts.ckks_vector_from(context, ew)
            decrypted_weights.append(torch.tensor(enc_vector.decrypt(),
dtype=torch.float32))
        except Exception as e:
            return None  # Return None to prevent further errors
    if not decrypted_weights:
        return None
    return decrypted_weights
# 🔄 Federated Learning Client with Gradient Encryption
class FLClient(fl.client.NumPyClient):
    def __init__(self, model, train_loader, test_features, test_labels, context):
        self.model = model.to(device)
        self.train_loader = train_loader
        self.test_features = test_features
```

```
        self.test_labels = test_labels
        self.context = context
        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam(self.model.parameters(), lr=0.001,
weight_decay=0.002)

        self.noise_levels = []
        self.accuracies = []
        self.privacy_budgets = []
        self.training_times = []

        # 🟩 Apply Differential Privacy (DP)
        self.privacy_engine = PrivacyEngine(secure_mode=False)
        self.model, self.optimizer, self.train_loader =
self.privacy_engine.make_private_with_epsilon(
            module=self.model,
            optimizer=self.optimizer,
            data_loader=self.train_loader,
            target_epsilon=5.0,
            target_delta=1e-5,
            epochs=5,
            max_grad_norm=2.0
        )

    def get_parameters(self, config):
        print(f"🔒 Client {client_idx} encrypting model parameters...")
        weights = [val.cpu().detach().numpy() for val in self.model.parameters()]
        return encrypt_weights(weights, self.context)  # Now it's a list of serialized
encrypted vectors

    def set_parameters(self, parameters):
        decrypted_params = decrypt_weights(parameters, self.context)

        if decrypted_params is None:
            return  # Prevents the crash

        for param, new_param in zip(self.model.parameters(), decrypted_params):
            param.data = new_param.view(param.shape).to(param.device)
    # 🟩 Modify fit function to Encrypt and Decrypt Gradients
    def fit(self, parameters, config):
        print(f"🔓 Client {client_idx} received encrypted parameters, starting
training...")

        self.set_parameters(parameters)  # Decrypt and load weights
        self.model.train()
        total_loss = 0
        correct = 0
        total_samples = 0
```

```
        round_loss = 0.0 # 🟩 Initialize round_loss before using it
        start_time = time.time()
        for epoch in range(5):
            running_loss = 0.0
            for batch_features, batch_labels in self.train_loader:
                batch_features, batch_labels = batch_features.to(device),
batch_labels.to(device)
                self.optimizer.zero_grad()
                outputs = self.model(batch_features)
                loss = self.criterion(outputs, batch_labels)
                loss.backward()
                encrypted_grads = []
                for param in self.model.parameters():
                    if param.grad is not None:
                        flattened_grad = param.grad.data.cpu().numpy().flatten()
                        encrypted_grads.append(ts.ckks_vector(self.context,
flattened_grad).serialize())

                self.optimizer.step() # Update model normally

                running_loss += loss.item()
                pred = outputs.argmax(dim=1, keepdim=True)
                correct += pred.eq(batch_labels.view_as(pred)).sum().item()
                total_samples += batch_labels.size(0)
            end_time = time.time()
            epoch_loss = running_loss / len(self.train_loader)
            round_loss += epoch_loss
            avg_loss = running_loss / len(self.train_loader)
            accuracy = 100.0 * correct / total_samples
            print(f"  Epoch {epoch+1}: Loss = {avg_loss:.4f}")

        print(f"🟩 Client {client_idx} training completed, encrypting updated model
weights...")
        self.noise_levels.append(selected_sigma)
        self.accuracies.append(accuracy)
        training_time = end_time - start_time
        self.training_times.append(training_time) # Store computation time
        # 🟩 Encrypt updated weights
        new_weights = [val.cpu().detach().numpy() for val in self.model.parameters()]
        return encrypt_weights(new_weights, self.context), len(self.train_loader.dataset),
{"loss":avg_loss,"accuracy":accuracy}
    def evaluate(self, parameters, config):
        print(f"🔍 Client {client_idx} evaluating model...")
        self.set_parameters(parameters)
        self.model.eval()
        with torch.no_grad():
            test_features = torch.tensor(self.test_features, dtype=torch.float32).to(device)
            test_labels = torch.tensor(self.test_labels, dtype=torch.long).to(device)
```

```
            outputs = self.model(test_features)
            y_pred = torch.argmax(outputs, dim=1).cpu().numpy()
            accuracy = accuracy_score(test_labels.cpu().numpy(), y_pred)
            loss = self.criterion(outputs, test_labels).item()
            print(f" Client {client_idx} Accuracy: {accuracy:.4f}")
        return float(loss), len(test_labels), {"accuracy": accuracy}

# 8⃣ Run the Federated Client
if _name_ == "_main_":
    train_loader = DataLoader(TensorDataset(
        torch.tensor(train_features, dtype=torch.float32),
        torch.tensor(train_labels, dtype=torch.long)
    ), batch_size=32, shuffle=True)

    model = MLPClassifier(input_dim=50, num_classes=len(set(train_labels)))
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f" Starting Client {client_idx}")
    client = FLClient(model, train_loader, test_features, test_labels, client_context)
    fl.client.start_numpy_client(server_address="127.0.0.1:9090", client=client)
```

Reference:

1. Ahmad Chaddad, Yihang Wu, Christian Desrosiers," Federated Learning for Healthcare Applications Publisher " , IEEE,2023.
2. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy preserving deep learning via additively homomorphic encryption ", IEEE,2022.
3. Daniel J. Beutel , Taner Topal ,Akhil Mathur , Xinchi Qiu ,Javier Fernandez-Marques ,Yan Gao ,Lorenzo Sani ,Kwing Hei Li , Titouan Parcollet , Pedro Porto Buarque de Gusm˜ao , Nicholas D. Lane "Flower: A Friendly Federated Learning Framework Publisher " arXiv,5 Mar 2022.
4. Zhiming Xia, Yang Chen,Chen Xu "Multiview PCA: A Methodology of Feature Extraction and Dimension Reduction for High-Order Data",IEEE Transactions on Cybernetics Volume: 52, Issue: 10, October 2022.
5. Rieke, N., Hancox, J., Li, W., Milletari, F., Roth, H. R., Albarqouni, S.,& Cardoso, M. J. "Federated Learning for Healthcare: Privacy-Preserving Data Sharing in Medical Research "Nature Medicine, 2020.
6. E. Bagdasaryan, B. S. Veeling, and M. Van Der Schaar, J. "Differential privacy for decentralized deep learning, "Mach. Learn. Res,vol. 21, no. 91, pp. 1–36, 2020.
7. Bian Zhu, Ling Niu" A privacy-preserving federated learning scheme with homomorphic encryption and edge computing" Science Direct Alexandria Engineering Journal Volume 118, pp 11-20,2025.
8. H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang" Learning differentially private recurrent language models, "arXiv:1710.06963, 2017.
9. L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai," Privacy preserving deep learning via additively homomorphic encryption"IEEE Trans. Inf. Forensics Security, vol. 13, no. 5, pp. 1333–1345, May 2018.
10. S. Truex, L. Liu, K.-H. Chow, M. E. Gursoy, and W. Wei," LDP-Fed: Federated learning with local differential privacy", Proc. 3rd ACM Int. Workshop Edge Syst., Anal. Netw, pp. 61–66, 2020.

11. Maria Iqbal, Asadullah Tariq, Muhammad Adnan, Irfan UD Din, Tariq Qayyum “FL-ODP: An Optimized Differential Privacy Enabled Privacy Preserving Federated Learning “,IEEE October 2023

12. Jaehyeok Lee 1, Phap Ngoc Duong 1,2 and Hanho Lee” . Configurable Encryption and Decryption Architectures for CKKS-Based Homomorphic Encryption”, NIH ,2023.

13. P. M. Mammen,” Federated learning: Opportunities and challenges”, arXiv:2101.05428, 2021.

14. Ismail Chahid, Aissa Kerkour Elmiad, Mohammed Badaoui “Data Preprocessing For Machine Learning Applications in Healthcare: A Review “, 14th International Conference on Intelligent Systems: Theories and Applications (SITA), 22-23 November 2023

15. K. Narmadha, P. Varalakshmi ,” Federated Learning in Healthcare: A Privacy Preserving Approach “IOS Press, Category :Research Article, pp 194-198,2022

16. Rana Ali, Asmaa Mohammed, Yara Yasser and Mostafa Hesham ,” A quantum convolutional network and ResNet (50)-based classification architecture for the MNIST medical dataset “ easychair, Apri2024

17. D. A. Tamburri ,” . Design principles for the general data protection regulation (GDPR): A formal concept analysis and its evaluation” ScienceDirect Inf. Syst., vol. 91, Jul. 2020, Art. no. 101469.

18. Pallavi Dhade, Prajakta Shirke,” Federated Learning for Healthcare: A Comprehensive Review” MDPI, Vol 59 10.3390,2024

19. Karthik Meduri ,Steven Brown , Snehal Satish , Hari Gonaygunta ,“Leveraging federated learning for privacy-preserving analysis of multi-institutional electronic health records in rare disease research “ScienceDirect, Journal of Enonomy and Technology,Vol 3,pp 177-189,2024

20. Chenhan Wang,” Federated Learning with ResNet-18 for Medical Image Diagnosis” 2023 8th International Conference on Multimedia Systems and Signal Processing 20 September 2023