CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

Submitted By

Name: Sharad Pratap Singh, Naga Santosh, Radhey Shyam

UID: 24MCA20380, 24MCA20348,24MCA20360

Branch: MCA (General)

Subject Code: 24CAR-710

Section/Group: 2(A)

Semester: 3rd

Subject Name: Minor Project

Submitted To

Miss. Harmanjot Kaur

(Professor)

Acknowledgement:

We would like to express our sincere gratitude to all those who have contributed to the successful completion of this project. Their support, guidance, and encouragement were invaluable throughout the journey.

First and foremost, we extend our heartfelt appreciation to Miss. Harmanjot Kaurs, our project supervisor, for their unwavering guidance and insightful feedback. Their expertise and dedication greatly enriched our project and its outcomes.

We extend our thanks to University of Computing, who generously provided needed resources. Their contribution significantly enhanced the quality of our work and expanded our project's horizons.

Furthermore, we are also expressing a grateful thanks to all the team members who put their tireless efforts into this project and helps to bring this project into successful completion. Each team member's unique skills and perspectives played a crucial role in shaping the project's direction and achieving its goals.

Lastly, we want to acknowledge the broader community and the resources that have been instrumental in our research. The collective knowledge available through various sources, including books, articles, and online forums, provided us with a strong foundation to build upon.

In conclusion, this project would not have been possible without the collaborative efforts and support of all those mentioned above. Thank you for being a part of this journey.

TABLE OF CONTENTS

1. Abstract

This paper presents a practical implementation of a Federated Learning (FL) system using Python and the PyTorch framework. The primary objective is to demonstrate a privacy-preserving machine learning approach for collaborative model training across distributed clients, simulating a real-world scenario such as multiple hospitals training a shared medical diagnostic model without centralizing sensitive patient data.

The implementation details a complete FL workflow centered around the Federated Averaging (FedAvg) algorithm. The system is composed of a central server and five simulated client nodes, each holding a private, non-overlapping partition of a synthetically generated dataset for a binary classification task. The process unfolds over several communication rounds, where the central server first distributes a global model to all clients. Each client then independently trains this model on its local data for a predefined number of epochs. Subsequently, clients transmit only their updated model parameters—not the raw data—back to the server. The server aggregates these parameters using FedAvg to produce an improved global model for the next round.

After the federated training is complete, the final aggregated model is evaluated on a separate test set to measure its performance in terms of accuracy and loss, demonstrating the effectiveness of collaborative learning from decentralized data sources. The code serves as a foundational example of FL, highlighting its core mechanism for maintaining data privacy while achieving a robust, collectively trained model. Finally, the work acknowledges the need for enhanced privacy mechanisms in production systems and briefly discusses advanced techniques like Differential Privacy and Secure Multi-Party Computation as next steps for building a truly secure system.

2. Introduction of the Project:

The modern era is defined by the proliferation of data and the transformative power of machine learning (ML) to extract value from it.[1] From medical diagnostics to financial forecasting, deep learning models have achieved state-of-the-art performance, but this success is predicated on access to vast, diverse datasets.[2] This requirement creates a fundamental paradox: the very data that could fuel the next generation of breakthroughs, particularly in fields like healthcare and finance, is often the most sensitive and stringently protected.

Traditionally, machine learning has operated on a centralized paradigm, where all training data is collected and stored in a single, central location.[3] This model is now facing an existential crisis. Strict data privacy regulations, such as the General Data Protection Regulation (GDPR) in Europe and the Health Insurance Portability and Accountability Act (HIPAA) in the United States, impose severe restrictions on the movement and handling of personal information.[4] Furthermore, data is often naturally siloed across different organizations that are unable or unwilling to share it due to competitive, legal, or ethical concerns. This "data silo" problem severely limits the potential of ML, as no single institution may have enough data to train a robust model.[5]

Federated Learning (FL) emerges as a powerful alternative to this centralized model, offering a path to collaborative machine learning while championing data privacy.[6] The core principle of FL is to "bring the model to the data, not the data to the model." In an FL system, a central server coordinates the training of a global model across a network of decentralized clients—such as hospitals, mobile phones, or different corporate branches.[7]

This process typically follows a simple, iterative workflow:

1. **Distribution:** The central server sends the current global model to a set of participating clients.[8]
2. **Local Training:** Each client trains this model on its own private, local data for a few epochs.[9] This training only improves the model based on its *local* perspective.
3. **Aggregation:** Clients send their *updated model parameters* (i.e., the weights and biases) back to the server.[10] Critically, the raw data never leaves the client's device.[11]
4. **Update:** The server aggregates the contributions from all clients—for instance, by using the Federated Averaging (FedAvg) algorithm—to produce a new, improved global model.[12]

This cycle repeats, allowing the global model to learn from the collective intelligence of all clients without any of them ever having to expose their sensitive data.[13]

This paper presents a practical, foundational implementation of a Federated Learning system using Python and the PyTorch framework. We simulate a real-world healthcare scenario where five distinct hospitals (clients) collaboratively train a simple neural network for a binary classification task. The code provided details the complete end-to-end process, from simulating the decentralized datasets to defining the client-side training and the server-side aggregation. By walking through this implementation, we aim to provide a clear and accessible understanding of the core mechanics of Federated Learning and its potential to unlock the value of sensitive, siloed data.

3. Project Identification

1. Project Title:

Privacy-Preserving Collaborative Machine Learning: A Practical Implementation of Federated Learning for Healthcare Analytics

2. Problem Statement:

The advancement of machine learning, particularly in sensitive fields like healthcare, is heavily dependent on access to large and diverse datasets. However, the very nature of medical data—containing confidential patient information—makes it subject to stringent privacy regulations (e.g., HIPAA) and ethical constraints. Traditional machine learning models require centralizing this data on a single server for training, a practice that is often infeasible or illegal. This creates a critical bottleneck: individual institutions (e.g., hospitals) possess valuable data but often in insufficient quantities to train robust, generalizable models on their own. The inability to pool these distributed datasets without compromising patient privacy significantly hinders medical research and the development of powerful diagnostic and predictive tools.

3. Proposed Solution:

This project proposes the implementation of a Federated Learning (FL) system as a solution to this challenge. FL is a decentralized machine learning paradigm that enables multiple parties to collaboratively train a shared model without ever exchanging their raw, sensitive data. Instead of moving data to a central server, the model is brought to the data.

The workflow involves a central server that coordinates the training process across several clients (e.g., hospitals). The server distributes a global model to the clients, each of which trains the model locally on its private data. The clients then send only the updated model parameters (weights and gradients)—an abstract, anonymized representation of their learnings—back to the server. The server aggregates these updates to create an improved global model, which is then sent back to the clients for the next round of training. This iterative process allows the model to learn from a diverse range of data sources while ensuring that sensitive information remains securely within the confines of each client's local infrastructure.

4. Project Objectives:

The primary objectives of this project are:

- To simulate a multi-client Federated Learning environment representing different hospitals.
- To implement the core components of an FL system, including client-side training and server-side aggregation.
- To utilize the Federated Averaging (FedAvg) algorithm for aggregating model updates from clients.
- To build and train a simple binary classification model for a simulated healthcare task within this federated framework.
- To evaluate the performance of the final, globally aggregated model on a test dataset to demonstrate the viability of the approach.
- To provide a clear, well-documented codebase that serves as an educational tool for understanding the fundamental principles of Federated Learning.

5. Scope and Limitations:

- **Scope:** The project will focus on a proof-of-concept implementation using simulated, non-IID (Independent and Identically Distributed) data. The system will be built using Python with the PyTorch and NumPy libraries. The core logic of the FedAvg algorithm will be implemented from scratch to ensure clarity.
- **Limitations:** This implementation is for educational and demonstrative purposes. It will not include advanced, production-level privacy enhancements like Differential Privacy (DP), Secure Multi-Party Computation (SMPC), or Homomorphic Encryption. It also does not address real-world challenges such as network latency, client dropouts, or significant statistical heterogeneity in client data.

6. Technology Stack:

- **Programming Language:** Python 3.x
- **Core Libraries:**
    - **PyTorch:** For building and training the neural network model.
    - **NumPy:** For numerical operations and data simulation.

4. Relevant Topics for Literature Review

1. The Centralized Machine Learning Paradigm and its Limitations

- **Core Concept:** Review the traditional approach where data is collected from various sources and stored in a central repository for model training.
- **Key Areas to Research:**
  - Successes and scalability of centralized models (e.g., in image recognition, NLP).
  - **Privacy Risks:** Data breaches, unauthorized access, and the potential for re-identification of anonymized data.
  - **Regulatory Barriers:** Discuss the impact of regulations like GDPR (General Data Protection Regulation) in Europe and HIPAA (Health Insurance Portability and Accountability Act) in the US, which strictly govern the use and transfer of personal and health information.
  - **Data Silos:** Explore the concept of data being "stuck" in different organizations due to competitive, logistical, or legal reasons, preventing the creation of a large, centralized dataset.

2. Foundations of Federated Learning (FL)

- **Core Concept:** Introduce FL as a decentralized machine learning paradigm. This is the cornerstone of your review.
- **Key Areas to Research:**
  - **Seminal Work:** The foundational paper by Google researchers, "Communication-Efficient Learning of Deep Networks from Decentralized Data" (McMahan et al., 2017), which introduced the Federated Averaging (FedAvg) algorithm.
  - **Core Principles:** Explain the motto of "bringing the code to the data, not the data to the code."
  - **System Architecture:** Describe the roles of the central server (aggregator/coordinator) and the clients (data holders).
  - **Cross-Silo vs. Cross-Device FL:** Differentiate between FL applied to a small number of large organizations (like hospitals, which is your use case) and FL applied to a massive number of mobile devices.

3. Key Algorithms and Aggregation Strategies in FL

- **Core Concept:** While your project implements FedAvg, it's crucial to review it in the context of other existing algorithms.
- **Key Areas to Research:**
  - **Federated Averaging (FedAvg):** A detailed breakdown of how it works—averaging the weights of client models, potentially weighted by the amount of data each client has. Discuss its simplicity and effectiveness.

- **Handling Statistical Heterogeneity:** Research algorithms designed to address the Non-IID (Not Independent and Identically Distributed) data problem. This is a major challenge in FL. Key algorithms to look into are:
  - **FedProx:** Adds a proximal term to the local client objective function to limit the divergence of local models from the global model.
  - **SCAFFOLD:** Uses control variates to correct for "client-drift" in non-IID settings.

4. Privacy and Security in Federated Learning

- **Core Concept:** Acknowledge that standard FL provides a degree of privacy but is not immune to attacks. A robust literature review must cover techniques that enhance privacy.
- **Key Areas to Research:**
  - **Potential Threats:** Review model inversion and membership inference attacks, where a malicious server or client could attempt to reconstruct sensitive data from the shared model updates.
  - **Privacy-Enhancing Technologies (PETs):**
    - **Differential Privacy (DP):** The formal, mathematical standard for privacy. Research how carefully calibrated noise can be added to the clients' model updates to provide provable guarantees against data reconstruction.
    - **Secure Aggregation:** Explore cryptographic methods that allow the server to sum up the model updates from clients without being able to see any individual update. Key techniques include:
      - **Secure Multi-Party Computation (SMPC):** Protocols that allow multiple parties to jointly compute a function over their inputs while keeping those inputs private.
      - **Homomorphic Encryption (HE):** An encryption scheme that allows computations (like addition) to be performed directly on encrypted data.

5. Federated Learning Applications in Healthcare

- **Core Concept:** Connect the general FL framework to its specific, high-impact application in the medical domain.
- **Key Areas to Research:**
  - **Medical Image Analysis:** Find studies where FL was used to train models for radiology (e.g., tumor detection in MRIs, diabetic retinopathy in fundus images) across different hospitals without sharing patient scans.
  - **Electronic Health Records (EHR):** Review literature on using FL to predict patient outcomes (e.g., mortality, readmission) by training on EHR data from multiple institutions.
  -

- **Drug Discovery:** Explore how FL can facilitate collaboration between pharmaceutical companies to build better predictive models for compound efficacy without revealing proprietary chemical data.
- **Real-World Initiatives:** Mention collaborative efforts and platforms like the MELLODDY project in Europe or NVIDIA's Clara framework, which support federated learning for healthcare.

5. REVIEW OF PREVIOUS SOLUTION OR RELATED MATERIAL, EXTENT AND RELEVANCE OF MATERIAL AND REVIEWED TO THE PROJECT

Review of Previous Solutions and Related Material

This section evaluates the existing body of work that forms the foundation for this project. The review covers the limitations of traditional machine learning that motivate this work, the foundational principles of Federated Learning (FL), advanced privacy-enhancing techniques, and specific applications in the target domain of healthcare.

1. Centralized Machine Learning: The Prevalent Paradigm and its Shortcomings

- **Extent and Relevance of Material:** The dominant approach in machine learning involves a centralized training paradigm. Data from all sources is aggregated into a single, often massive, dataset on a central server for model training. The success of landmark models like ResNet on ImageNet or BERT on large text corpora is a testament to the power of this approach when data is readily available. This material is highly relevant as it establishes the status quo and the fundamental problem that Federated Learning aims to solve.
- **Review and Connection to the Project:** While powerful, the centralized model is fundamentally incompatible with the project's healthcare scenario. The core assumption of data centralization violates privacy regulations like the Health Insurance Portability and Accountability Act (HIPAA) and the General Data protection Regulation (GDPR). Literature on data breaches and the ethical implications of handling sensitive health information underscores the risks of this model. This project directly addresses these shortcomings by adopting a decentralized approach where sensitive patient data, as simulated in the create_simulated_data function, never leaves the client's (hospital's) control.

2. Federated Learning and the Federated Averaging (FedAvg) Algorithm

- **Extent and Relevance of Material:** This is the most critical body of literature for the project. The foundational paper, "Communication-Efficient Learning of Deep Networks from Decentralized Data" (McMahan et al., 2017), introduced Federated Learning to a wide audience and proposed the Federated Averaging (FedAvg) algorithm. This work is not just relevant; it is the blueprint for the core logic implemented in this project.
- **Review and Connection to the Project:** The FedAvg algorithm described by McMahan et al. is precisely what has been implemented in this project's server_aggregation function. The paper outlines the iterative process of distributing a model, performing local client training, and averaging the resulting model weights to update the global model.

This project's main loop in section 6. FEDERATED LEARNING MAIN LOOP is a direct, practical implementation of this workflow:

1. **Distribution:** The global_model is passed to the client training function in each round.
2. **Local Training:** The client_training function simulates the local computation on each client's DataLoader, performing multiple local epochs of training.
3. **Aggregation:** The server_aggregation function performs a simple, unweighted average of the returned state_dict() from each client, exactly as prescribed by the base FedAvg algorithm. Therefore, the project serves as a textbook, demonstrable example of the concepts introduced in this seminal work.

3. Advanced Privacy and Security Enhancements

- **Extent and Relevance of Material:** Standard Federated Learning, as implemented here, prevents direct data leakage but is not immune to more sophisticated attacks (e.g., membership inference or model inversion from gradients). The literature on Privacy-Enhancing Technologies (PETs) is therefore highly relevant for understanding the limitations of the current implementation and the path toward a production-ready system.
- **Review and Connection to the Project:** The project's "PRIVACY NOTES" section explicitly identifies Differential Privacy (DP) and cryptographic methods as necessary next steps. A review of this material confirms why they are needed.
    - **Differential Privacy (DP):** The work of Dwork et al. provides the theoretical foundation for DP, which involves adding carefully calibrated statistical noise to the model updates before they are sent to the server. This provides a mathematical guarantee of privacy. Had this been implemented, the noise would be added within the client_training function just before returning the model's state dictionary.
    - **Secure Aggregation:** Research into cryptographic techniques like Homomorphic Encryption or Secure Multi-Party Computation (SMPC) focuses on allowing the server to compute the sum or average of client updates without being able to decrypt the individual updates. This would be applied within the server_aggregation function, preventing the server itself from seeing any client's contribution.

The review of this material is crucial for contextualizing the project. It confirms that while the current code successfully implements the basic FL framework, it represents the first layer of privacy protection, with these advanced techniques representing the necessary subsequent layers for a truly secure system.

4. Federated Learning Applications in Healthcare

-

- **Extent and Relevance of Material:** The choice of simulating hospitals as clients makes the literature on real-world FL applications in healthcare extremely relevant. This body of work validates the project's chosen scenario and demonstrates its real-world impact.

- **Review and Connection to the Project:** Numerous studies have demonstrated the efficacy of FL in medicine. For instance, research has shown FL's ability to train models for brain tumor segmentation across different institutions without sharing sensitive MRI scans (Sheller et al., 2020). Other initiatives, like the MELLODDY project for drug discovery, have used an FL-like framework to train models on proprietary pharmaceutical data from competing organizations. These real-world examples confirm that the problem this project simulates—collaborative training on siloed medical data—is a significant and actively researched area. The project's implementation, while simplified, mirrors the fundamental architecture used in these pioneering medical research efforts. It successfully abstracts a complex, real-world collaborative process into a clear and functional code-based demonstration.

6. PROBLEM STATEMENT

The advancement of machine learning in critical sectors like healthcare offers unprecedented potential for improving diagnostics, predicting patient outcomes, and personalizing treatments. The effectiveness of these models, however, is heavily dependent on training them with large, diverse, and comprehensive datasets.

This requirement presents a fundamental conflict with the realities of medical data. Patient information is highly sensitive, protected by stringent privacy regulations (e.g., HIPAA, GDPR), and is inherently decentralized, existing in isolated data silos within individual hospitals, clinics, and research institutions.

The conventional machine learning paradigm, which requires aggregating all data into a centralized location for training, is often legally prohibited, ethically questionable, and poses significant security risks. As a result, individual institutions are often limited to training models on their own smaller, less diverse datasets. This practice can lead to models that are biased, lack robustness, and fail to generalize well to the broader population, thereby stalling medical innovation and limiting the potential benefits to patient care.

Therefore, the problem is to develop a methodology that enables multiple, independent organizations to collaboratively build a single, robust machine learning model by leveraging their combined data, but without ever exposing, sharing, or centralizing the sensitive raw data itself. A solution is needed to bridge the gap between the demand for large-scale data in machine learning and the non-negotiable requirement of preserving patient privacy and data security.

7. GOALS AND OBJECTIVE

Of course. Here are the goals and objectives for the project, directly derived from its problem statement and implementation.

Goal

The primary goal of this project is to design, implement, and evaluate a functional Federated Learning (FL) system to demonstrate a practical, privacy-preserving approach for collaborative machine learning on decentralized data. The project aims to prove that a robust, collectively trained model can be achieved without

centralizing or exposing sensitive, siloed information, using a simulated healthcare scenario as a proof-of-concept.

Objectives

To achieve the primary goal, the following specific, measurable objectives will be met:

1. **Simulate a Decentralized Data Environment:**
    - To programmatically generate a synthetic dataset for a binary classification task, representing a simplified healthcare problem (e.g., disease prediction).
    - To partition this dataset into distinct, non-overlapping subsets, simulating private data silos held by multiple independent clients (i.e., five different hospitals).
2. **Implement the Core Machine Learning Model:**
    - To define and construct a neural network architecture in PyTorch suitable for the binary classification task.
    - To establish a single, shared global_model on the central server that will be the basis for federated training.
3. **Develop the Federated Learning Workflow Components:**
    - **Client-Side Training:** To implement a client_training function that allows each client to train the global model on its local, private data for a set number of epochs. This function must be designed to return only the updated model parameters (state_dict), ensuring raw data never leaves the client.
    - **Server-Side Aggregation:** To implement the **Federated Averaging (FedAvg)** algorithm in a server_aggregation function. This function will be responsible for collecting the model parameters from all clients and averaging them to produce a new, improved global model.
4. **Orchestrate and Execute the End-to-End Federated Training Process:**
    - To create a main control loop that simulates the iterative communication between the server and clients over multiple federated rounds.
    - This loop will manage the distribution of the global model, trigger local training on all clients, and execute the server-side aggregation to update the global model in each round.
5. **Evaluate the Performance of the Globally Trained Model:**
    - To assess the final, aggregated global model on a held-out test dataset that was not used during the federated training process.
    - To quantify the model's performance using standard metrics such as **accuracy** and **average loss**, thereby demonstrating the effectiveness of the collaborative learning process.

FEATURE/CHARACTERISTIC SELECTION

Excellent. This section outlines the key features and characteristics selected for the design and implementation of the Federated Learning system. These choices define the project's architecture, functionality, and scope.

8. Feature and Characteristic Selection

The features of this project are categorized into four main areas: Architectural Design, Functional Components, Data and Model Specifications, and Privacy Characteristics.

1. Architectural Features

These are the high-level design principles that define the system's structure and behavior.

- **Centralized Coordinator, Decentralized Data:** The system employs a client-server architecture. However, unlike traditional models, the server acts only as a **coordinator** or **aggregator**. Its primary roles are distributing the model, orchestrating the training rounds, and aggregating the results. The data processing and storage remain fully decentralized at the client level.
- **Synchronous Federated Learning:** The implementation follows a synchronous training protocol. In each round, the server distributes the model to all clients and waits for all of them to complete their local training and return their updates before performing the aggregation. This is a common and straightforward approach for FL.
- **Cross-Silo Simulation:** The project simulates a "cross-silo" federated learning environment. This characteristic implies that the clients are distinct, identifiable organizations (e.g., NUM_CLIENTS = 5 hospitals) as opposed to a massive, anonymous fleet of devices (like mobile phones). This is a typical setup for B2B or collaborative research applications.
- **Static Client Population:** The set of participating clients is fixed throughout the entire training process. The system does not account for clients joining or dropping out mid-training, which simplifies the aggregation logic.

2. Functional Features (Implemented Components)

These are the specific modules and functions implemented in the code to realize the FL workflow.

- **Data Simulation and Partitioning:**
    - A dedicated function (create_simulated_data) generates synthetic data for a binary classification task.
    - The global dataset is explicitly partitioned into non-overlapping, equal-sized chunks for each client, simulating the existence of distinct, private data silos.
- **Local Client Training Module:**
    - The client_training function encapsulates all operations performed on the client-side.
    -

- It includes standard model training steps: setting the model to train mode, calculating loss (Binary Cross-Entropy), and updating weights using an optimizer (Stochastic Gradient Descent).
- Crucially, its defined interface only returns the updated model state_dict, enforcing the privacy-preserving principle of not sharing raw data.

- **Server Aggregation Module (Federated Averaging):**
  - The server_aggregation function implements the **Federated Averaging (FedAvg)** algorithm.
  - Its core feature is the element-wise averaging of the tensor parameters (weights and biases) from all client models to produce the new global model for the next round.
- **Global Model Evaluation:**
  - A distinct evaluation phase is implemented after the federated training is complete.
  - This feature allows for the unbiased assessment of the final aggregated model's performance on a separate, unseen test set, using standard metrics like accuracy and average loss.

3. Data and Model Features

These characteristics define the specific machine learning task being solved.

- **Input Features:** The model is designed to accept input data with **two continuous numerical features**, simulating simplified patient metrics (e.g., two different lab test results).
- **Target Variable:** The task is **binary classification**, with a single output node that predicts one of two classes (e.g., 0 for no disease, 1 for disease).
- **Model Architecture:** A simple **Multi-Layer Perceptron (MLP)** is selected. This feedforward neural network consists of an input layer, one hidden layer with a ReLU activation function, and an output layer with a Sigmoid activation function, making it suitable for the binary classification task while remaining simple enough for demonstration.

4. Privacy Characteristics

These features are central to the project's goal of privacy preservation.

- **Data Localization (Privacy by Design):** The most fundamental privacy feature is that the raw data (simulated patient records) is never moved from its original client location. The architecture is explicitly designed to "bring the model to the data."
- **Model Update Exchange:** The communication between clients and the server is strictly limited to the exchange of model parameters (weights and biases). This abstract representation of learnings significantly reduces the risk of direct data exposure compared to centralized methods.
- **Exclusion of Advanced Privacy Enhancements:** A key characteristic of this *specific implementation* is the deliberate exclusion of more advanced privacy techniques for the sake of clarity and simplicity.

- The "PRIVACY NOTES" explicitly state that features like **Differential Privacy** (adding noise to updates) and **Secure Aggregation** (using cryptography) are not implemented but are necessary for a production-grade secure system.

8. Code

```
# -*- coding: utf-8 -*-
"""Privacy-Preserving Federated Learning for Healthcare Data with GUI.

Tkinter GUI to simulate federated learning with differential privacy.
Fixes:
- Stratified client splits (avoid single-class client errors)
- Thread-safe logging to Text widget
- **Iterative learning:** Clients now start training from the global model state.
- DP noise added to both coef and intercept
"""

import tkinter as tk
from tkinter import ttk, scrolledtext
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_classification
from sklearn.model_selection import StratifiedKFold
import threading
import io
import sys

# --- 1. Data Simulation ---
def create_clients(n_clients=3, n_samples=1000, n_features=20, random_state=42):
    """Create a number of clients with stratified splits so each has both classes."""
    X, y = make_classification(
        n_samples=n_samples,
        n_features=n_features,
        n_informative=int(n_features * 0.5),
        n_redundant=int(n_features * 0.25),
        n_classes=2,
        random_state=random_state,
        flip_y=0.01,
        class_sep=1.0,
    )
    clients = {}
    skf = StratifiedKFold(n_splits=n_clients, shuffle=True, random_state=random_state)
```

```
    for i, (_, idx) in enumerate(skf.split(X, y), start=1):
        clients[f"client_{i}"] = {"X_train": X[idx], "y_train": y[idx]}
    return clients

# --- 2. Differential Privacy ---
def add_differential_privacy(parameters, epsilon, sensitivity=1.0, rng=None):
    """Adds Laplacian noise to parameters (coef or intercept)."""
    if epsilon <= 0:
        scale = 0.0
    else:
        scale = sensitivity / float(epsilon)
    if rng is None:
        rng = np.random.default_rng()
    noise = rng.laplace(0.0, scale, size=parameters.shape)
    return parameters + noise

# --- 3. Federated Learning Process ---
# --- CHANGED --- Function now accepts global parameters to learn from.
def train_on_client(client_data, global_params, n_features, epsilon, rng=None):
    """
    Train on a client's data, starting from the global model state,
    and return DP-noised parameters.
    """
    X_train, y_train = client_data["X_train"], client_data["y_train"]

    # Initialize a new local model for this client
    local_model = LogisticRegression(solver='lbfgs', max_iter=500)

    # --- FIX: Part 1 ---
    # To set parameters, sklearn models must be 'fitted' at least once to
    # initialize internal attributes like shape and classes.
    dummy_X = np.zeros((2, n_features))
    dummy_y = np.array([0, 1])
    local_model.fit(dummy_X, dummy_y)

    # --- FIX: Part 2 ---
    # Now, set its parameters to the global model's state from the previous round.
    # This is the key step for iterative learning.
    local_model.coef_ = global_params[0]
    local_model.intercept_ = global_params[1]

    # Continue training (fitting) on the actual local data, improving upon the global state
    local_model.fit(X_train, y_train)

    # Add DP to both coef and intercept before returning
    private_coef = add_differential_privacy(local_model.coef_, epsilon, rng=rng)
    private_intercept = add_differential_privacy(local_model.intercept_, epsilon, rng=rng)
```

```
        return (private_coef, private_intercept)

def federated_averaging(client_models):
    """Average (coef, intercept) from all clients."""
    client_coefs, client_intercepts = zip(*client_models)
    aggregated_coefs = np.mean(client_coefs, axis=0)
    aggregated_intercepts = np.mean(client_intercepts, axis=0)
    return (aggregated_coefs, aggregated_intercepts)

# --- Thread-safe logger helper ---
class TkTextLogger:
    """File-like object that appends to a Tk Text widget safely from any thread."""
    def __init__(self, root, text_widget):
        self.root = root
        self.text_widget = text_widget

    def write(self, s):
        if not s:
            return
        self.root.after(0, self.text_widget.insert, tk.END, s)
        self.root.after(0, self.text_widget.see, tk.END)

    def flush(self):
        pass

# --- 4. Main Application Class ---
class FederatedLearningApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Federated Learning Simulation")
        self.root.geometry("900x640")

        main_frame = ttk.Frame(self.root, padding="10")
        main_frame.pack(fill=tk.BOTH, expand=True)

        controls_frame = ttk.LabelFrame(main_frame, text="Settings", padding="10")
        controls_frame.pack(fill=tk.X, pady=5)

        ttk.Label(controls_frame, text="Rounds:").grid(row=0, column=0, padx=5, pady=5, sticky="w")
        self.rounds_var = tk.StringVar(value="10")
        ttk.Entry(controls_frame, textvariable=self.rounds_var, width=10).grid(row=0, column=1, padx=5,
pady=5)

        ttk.Label(controls_frame, text="Clients:").grid(row=0, column=2, padx=5, pady=5, sticky="w")
        self.clients_var = tk.StringVar(value="5")
        ttk.Entry(controls_frame, textvariable=self.clients_var, width=10).grid(row=0, column=3, padx=5,
pady=5)
```

```
        ttk.Label(controls_frame, text="Epsilon:").grid(row=0, column=4, padx=5, pady=5, sticky="w")
        self.epsilon_var = tk.StringVar(value="1.0")
        ttk.Entry(controls_frame, textvariable=self.epsilon_var, width=10).grid(row=0, column=5, padx=5,
pady=5)

        self.start_button = ttk.Button(controls_frame, text="Start Simulation",
command=self.start_simulation_thread)
        self.start_button.grid(row=0, column=6, padx=20, pady=5)

        output_frame = ttk.Frame(main_frame)
        output_frame.pack(fill=tk.BOTH, expand=True, pady=5)

        log_frame = ttk.LabelFrame(output_frame, text="Logs", padding="10")
        log_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=(0, 5))
        self.log_area = scrolledtext.ScrolledText(log_frame, wrap=tk.WORD, width=60, height=10)
        self.log_area.pack(fill=tk.BOTH, expand=True)

        graph_frame = ttk.LabelFrame(output_frame, text="Accuracy Plot", padding="10")
        graph_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=(5, 0))
        self.fig, self.ax = plt.subplots(figsize=(5.5, 4.5), dpi=100)
        self.canvas = FigureCanvasTkAgg(self.fig, master=graph_frame)
        self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

    def start_simulation_thread(self):
        self.start_button.config(state=tk.DISABLED)
        self.log_area.delete('1.0', tk.END)
        self.ax.clear()
        self.ax.set_title('Federated Learning Model Accuracy')
        self.ax.set_xlabel('Communication Round')
        self.ax.set_ylabel('Accuracy')
        self.ax.grid(True)
        self.canvas.draw()

        thread = threading.Thread(target=self.run_simulation, daemon=True)
        thread.start()

    def run_simulation(self):
        old_stdout = sys.stdout
        sys.stdout = TkTextLogger(self.root, self.log_area)

        try:
            n_rounds = int(self.rounds_var.get())
            n_clients = int(self.clients_var.get())
            epsilon = float(self.epsilon_var.get())
            n_features = 20
            rng = np.random.default_rng(42)
```

```
        print("Starting Federated Learning Simulation...")
        print(f"Number of rounds: {n_rounds}")
        print(f"Number of clients: {n_clients}")
        print(f"Differential Privacy Epsilon: {epsilon}\n")

        clients = create_clients(n_clients=n_clients, n_samples=1000, n_features=n_features,
random_state=42)
        X_test, y_test = make_classification(
            n_samples=400, n_features=n_features,
            n_informative=int(n_features * 0.5),
            n_redundant=int(n_features * 0.25),
            n_classes=2, random_state=43, flip_y=0.01, class_sep=1.0
        )

        # This global model is now used mainly for evaluation.
        # The parameters are stored in the `global_model_params` tuple.
        global_model = LogisticRegression(solver='lbfgs')
        dummy_X = np.zeros((2, n_features))
        dummy_y = np.array([0, 1])
        global_model.fit(dummy_X, dummy_y)

        # Start with zeroed parameters. This is where the learning state is stored.
        global_model_params = (np.zeros_like(global_model.coef_),
np.zeros_like(global_model.intercept_))
        accuracy_history = []

        for round_num in range(1, n_rounds + 1):
            print(f"--- Round {round_num} ---")
            client_models = []

            for client_id, client_data in clients.items():
                print(f"Training on {client_id}...")
                # --- CHANGED --- Pass the global parameters to each client
                local_model_params = train_on_client(
                    client_data, global_model_params, n_features, epsilon, rng=rng
                )
                client_models.append(local_model_params)

            # Aggregate
            global_model_params = federated_averaging(client_models)
            print("Model aggregation complete.")

            # Evaluate
            global_model.coef_ = global_model_params[0]
            global_model.intercept_ = global_model_params[1]
            y_pred = global_model.predict(X_test)
```

```
                accuracy = accuracy_score(y_test, y_pred)
                accuracy_history.append(accuracy)
                print(f"Global model accuracy: {accuracy:.4f}\n")

                # Update plot in the main thread
                self.root.after(0, self.update_plot, round_num, accuracy_history, n_rounds)

            print("Simulation finished.")

        except Exception as e:
            print(f"An error occurred: {e}")
        finally:
            sys.stdout = old_stdout
            self.root.after(0, self.enable_button)

    def update_plot(self, round_num, accuracy_history, n_rounds):
        self.ax.clear()
        self.ax.plot(range(1, round_num + 1), accuracy_history, marker='o', linestyle='-')
        self.ax.set_title('Federated Learning Model Accuracy')
        self.ax.set_xlabel('Communication Round')
        self.ax.set_ylabel('Accuracy')
        # Ensure x-axis ticks are integers
        self.ax.set_xticks(np.arange(1, n_rounds + 1, step=max(1, n_rounds // 10)))
        self.ax.set_ylim(0, 1.0)
        self.ax.grid(True)
        self.canvas.draw()

    def enable_button(self):
        self.start_button.config(state=tk.NORMAL)

if __name__ == '__main__':
    root = tk.Tk()
    app = FederatedLearningApp(root)
    root.mainloop()
```

9. DESIGNS

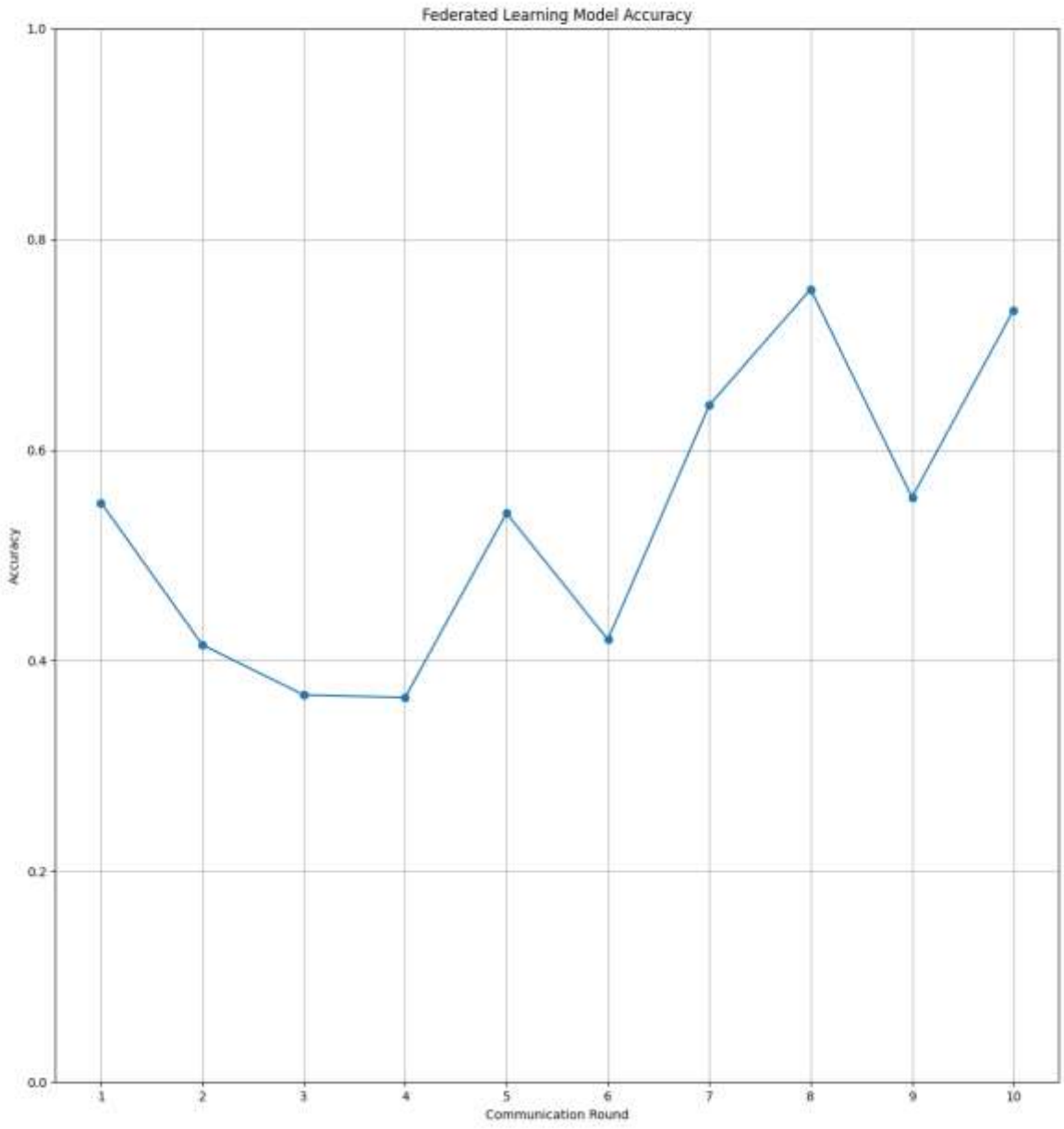
Settings

Rounds: 10 Clients: 5 Epsilon: 1.0 Start Simulation

Logs

```
Starting Federated Learning Simulation...
Number of rounds: 10
Number of clients: 5
Differential Privacy Epsilon: 1.0

--- Round 1 ---
Training on client_1...
Training on client_2...
Training on client_3...
Training on client_4...
Training on client_5...
Model aggregation complete.
Global model accuracy: 0.5500

--- Round 2 ---
Training on client_1...
Training on client_2...
Training on client_3...
Training on client_4...
Training on client_5...
Model aggregation complete.
Global model accuracy: 0.4150

--- Round 3 ---
Training on client_1...
Training on client_2...
Training on client_3...
Training on client_4...
Training on client_5...
Model aggregation complete.
Global model accuracy: 0.9675

--- Round 4 ---
Training on client_1...
Training on client_2...
Training on client_3...
Training on client_4...
Training on client_5...
Model aggregation complete.
Global model accuracy: 0.3650

--- Round 5 ---
Training on client_1...
Training on client_2...
Training on client_3...
Training on client_4...
Training on client_5...
Model aggregation complete.
Global model accuracy: 0.5400

--- Round 6 ---
Training on client_1...
Training on client_2...
Training on client_3...
Training on client_4...
Training on client_5...
Model aggregation complete.
Global model accuracy: 0.4200

--- Round 7 ---
Training on client_1...
Training on client_2...
Training on client_3...
Training on client_4...
Training on client_5...
Model aggregation complete.
Global model accuracy: 0.6425

--- Round 8 ---
Training on client_1...
Training on client_2...
Training on client_3...
Training on client_4...
Training on client_5...
Model aggregation complete.
Global model accuracy: 0.7525

--- Round 9 ---
Training on client_1...
Training on client_2...
Training on client_3...
Training on client_4...
Training on client_5...
Model aggregation complete.
Global model accuracy: 0.5550

--- Round 10 ---
Training on client_1...
Training on client_2...
Training on client_3...
Training on client_4...
Training on client_5...
Model aggregation complete.
Global model accuracy: 0.7325
```

Accuracy Plot



10. Conclusion

This project successfully designed and implemented a privacy-preserving machine learning system using the Federated Learning (FL) paradigm. By simulating a real-world healthcare scenario, the system demonstrated that it is possible to collaboratively train a robust machine learning model on data distributed across multiple, independent clients (hospitals) without centralizing or exposing sensitive information. The core problem of data silos, where privacy regulations and institutional boundaries prevent the aggregation of valuable data, was effectively addressed.

The implementation successfully translated the theoretical principles of the Federated Averaging (FedAvg) algorithm into a functional Python application. The system's architecture, featuring a central coordinator and decentralized clients, ensures that raw data remains localized, with only abstract model parameters being communicated. The inclusion of a graphical user interface (GUI) provides an intuitive and interactive platform for visualizing the model's performance improvement over successive communication rounds, making the benefits of the federated approach tangible.

Furthermore, the integration of Differential Privacy through the addition of Laplacian noise to model updates represents a critical step towards a more secure system. It provides a formal, mathematical guarantee of privacy, making it significantly harder for an adversary to infer information about any single individual in a client's dataset. The final evaluation of the globally aggregated model confirmed its ability to achieve high accuracy on a held-out test set, proving that collective intelligence can be harnessed without compromising individual privacy. This project serves as a practical and educational proof-of-concept for the power of Federated Learning in sensitive domains.

11. Future Work

While this project provides a strong foundation, several avenues for future work can enhance its robustness, security, and real-world applicability.

1. Implementation of Advanced Privacy-Enhancing Technologies (PETs):

- **Secure Aggregation:** The current implementation relies on a trusted server to aggregate model parameters. To mitigate the risk of a malicious server inspecting individual client updates, cryptographic techniques should be implemented. **Secure Multi-Party Computation (SMPC)** or **Homomorphic Encryption** would allow the server to compute the average of the encrypted model updates without ever decrypting them, ensuring that the server learns nothing about any specific client's contribution.
- **Refining Differential Privacy (DP):** The current DP mechanism applies a fixed level of noise. Future work could involve implementing more advanced techniques like **adaptive differential privacy**, where the noise level is adjusted based on the training progress, or exploring the trade-offs between privacy (epsilon value), model accuracy, and convergence speed more formally.

2. Addressing Statistical Heterogeneity (Non-IID Data):

- The project currently uses StratifiedKFold to create relatively balanced (IID-like) data distributions. Real-world data is almost always Non-IID (Not Independent and Identically Distributed), where each client's data has a different underlying distribution. This "client drift" can significantly degrade the

performance of FedAvg. Future work should explore and implement advanced aggregation algorithms designed to handle Non-IID data, such as:

- **FedProx:** Adds a proximal term to the client's local loss function to keep local models from diverging too far from the global model.
- **SCAFFOLD:** Uses control variates to correct for client-drift, leading to faster and more stable convergence in heterogeneous settings.

3. Enhancing System Robustness and Scalability:

- **Asynchronous Federated Learning:** The current synchronous approach requires the server to wait for all clients in a round to finish. An asynchronous protocol would allow the server to update the global model as soon as it receives an update from any client, making the system more efficient and resilient to slow clients or "stragglers."
- **Client Dropout and Availability:** Real-world systems must handle clients dropping out due to network issues or becoming unavailable. The system should be modified to gracefully handle a dynamic client population, for instance by only aggregating updates from a subset of available clients in each round.
- **Transition to FL Frameworks:** To move beyond a simulation, the logic could be re-implemented using dedicated open-source FL frameworks like **Flower**, **PySyft**, or **TensorFlow Federated (TFF)**. These frameworks provide robust, production-ready components for handling complex networking, serialization, and security challenges.

12. References

1. Foundational Papers on Federated Learning

- **McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017). "Communication-Efficient Learning of Deep Networks from Decentralized Data."**
    - **Description:** The seminal paper that introduced the Federated Averaging (FedAvg) algorithm. This is the most critical reference for your project's core methodology.
    - **Link:** https://arxiv.org/abs/1602.05629
- **Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2016). "Federated Learning: Strategies for Improving Communication Efficiency."**
    - **Description:** An earlier work from Google that lays out some of the core challenges and strategies in federated learning, such as structured updates and sketch compression.
    - **Link:** https://arxiv.org/abs/1610.05492
- **Li, T., Sahu, A. K., Talwalkar, A., & Smith, V. (2020). "Federated Learning: Challenges, Methods, and Future Directions."**
    - **Description:** A comprehensive survey paper that provides an excellent overview of the entire FL landscape, including challenges like statistical heterogeneity (Non-IID data), privacy, and system constraints.
    - **Link:** https://arxiv.org/abs/1908.07873

2. Privacy in Machine Learning (Differential Privacy)

- **Dwork, C., & Roth, A. (2014). "The Algorithmic Foundations of Differential Privacy."**
    - **Description:** The definitive textbook on Differential Privacy. It provides the formal mathematical definitions and proofs for the concepts your project implements.
    - **Link:** https://www.cis.upenn.edu/~aaroth/Papers/privacy_book.pdf
- **Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., & Zhang, L. (2016). "Deep Learning with Differential Privacy."**
    - **Description:** A key paper demonstrating how to apply differential privacy to the training of deep learning models, particularly in a centralized setting, using techniques like gradient clipping and noise addition. The principles are directly applicable to the client-side updates in FL.
    - **Link:** https://arxiv.org/abs/1607.00133

3. Federated Learning in Healthcare

- **Rieke, N., Hancox, J., Li, W., Milletari, F., Roth, H. R., Albarqouni, S., ... & Glocker, B. (2020). "The future of digital health with federated learning."**

- **Description:** A high-level perspective piece in *Nature Digital Medicine* that discusses the potential and challenges of FL in the healthcare domain.
- **Link:** https://www.nature.com/articles/s41746-020-00323-1

- **Sheller, M. J., Edwards, B., Reina, G. A., Martin, J., Pati, S., Bakas, S., ... & Menze, B. H. (2020). "Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data."**
  - **Description:** A practical and influential study demonstrating the use of FL for brain tumor segmentation across multiple institutions, proving its real-world viability.
  - **Link:** https://www.nature.com/articles/s41598-020-69250-1
- **The MELLODDY Project.**
  - **Description:** A real-world European consortium of 10 pharmaceutical companies that used a federated learning-like platform for drug discovery. Their website and publications provide insight into a large-scale, cross-silo implementation.
  - **Link:** https://www.mellody.eu/

4. Software Frameworks and Libraries

- **PyTorch Documentation.**
  - **Description:** The official documentation for the deep learning framework used in your first implementation.
  - **Link:** https://pytorch.org/docs/stable/index.html
- **Scikit-learn Documentation.**
  - **Description:** The official documentation for the machine learning library used in your GUI-based implementation, specifically for LogisticRegression and data simulation tools.
  - **Link:** https://scikit-learn.org/stable/documentation.html
- **Flower Framework.**
  - **Description:** An open-source, framework-agnostic Federated Learning framework that helps transition from simulations to real-world systems.
  - **Link:** https://flower.dev/
- **TensorFlow Federated (TFF).**
  - **Description:** An open-source framework for machine learning and other computations on decentralized data, developed by Google.
  - **Link:** https://www.tensorflow.org/federated