Project plan

Federated Learning for Disease Prediction - ARMY

Overall Project Brief:

Our final year project is to create a secure and practical **Federated Learning system for disease prediction** across three different hospitals. The project's core strength lies in its use of advanced **privacy-enhancing techniques**, such as **Homomorphic Encryption** or **Differential Privacy**, to ensure patient data remains confidential. The entire system will be accessible through a user-friendly web interface, demonstrating a real-world application of this cutting-edge technology.

The project is comprised of three main components:

- **Central Server:** This acts as the project's brain. It manages the **global model**, aggregates updates from all the hospital clients, and hosts a dashboard for real-time visualization of the training process.
- **Hospital Portals:** Each of the three hospitals will have a secure, unique web portal with two types of user logins:
    - **Admin Login:** For technical users to upload their hospital's private dataset (CSV/Excel), trigger the local training process, and monitor the training status.
    - **Doctor Login:** For end-users to input single-patient data and receive real-time disease predictions from the globally trained model.
- **Privacy Layer:** This is the project's key differentiator. The system will implement either **Homomorphic Encryption** (performing computations on encrypted data) or **Differential Privacy** (adding noise to data) to prevent privacy leaks and enhance security.

This project is a comprehensive showcase of skills in machine learning, data privacy, and web development, addressing a critical challenge in collaborative healthcare research.

Federated Learning Workflow

This workflow outlines the cyclical process of how the central server and hospital clients interact to train the disease prediction model without sharing any raw patient data.

1. Initialization Phase

- **Central Server:** An initial **global model** (e.g., a neural network) is created with random weights.
- **Central Server & Hospital Portals:** A hospital admin logs into their portal and uploads their private patient dataset (CSV or Excel). The central server then sends a copy of the initial global model to the local server of each hospital.

2. Local Training Phase

- **Hospital Admin Portal:** The admin triggers the training process by clicking a "Start Training" button.
- **Clients:** Each hospital's local server trains the received model on its unique, private patient dataset.
- **Privacy Enhancement:**
  - **Homomorphic Encryption (HE):** The hospital client **encrypts** the updated model weights using a public key provided by the central server.
  - **Differential Privacy (DP):** The hospital client adds a carefully calibrated amount of **random noise** to the updated model weights.
- **Status Update:** The admin portal provides a real-time status update, showing local model accuracy, training loss, and progress. The privacy-enhanced updates are then sent to the central server.

3. Aggregation Phase

- **Central Server:** The server receives the privacy-enhanced model updates from all three hospitals.
- **Aggregation:** The server aggregates these updates (e.g., by computing a weighted average). If using HE, this is done directly on the encrypted data.
- **Dashboard Update:** The central server's dashboard is updated with the performance metrics of the new global model, showcasing the progress of the training.
-

4. Prediction Phase

- **Doctor's Portal:** A doctor logs in and enters a single patient's data into a form.
- **Local Prediction:** The hospital's local server uses the **latest global model** to make a prediction on the single patient's data. The raw patient data **never leaves the hospital**.
- **Result Display:** The prediction is displayed instantly on the doctor's screen.

This cyclical process repeats, allowing the global model to continuously improve with each training round, all while maintaining the privacy of every patient's data.

Required Tools & Technologies

Core Federated Learning Frameworks

- **PySyft**: A powerful Python library built on PyTorch that's excellent for secure and private federated learning. It's especially useful for implementing **Homomorphic Encryption** because of its strong focus on secure multi-party computation.
- **TensorFlow Federated (TFF)**: A robust framework from Google. It's a great choice if you're using TensorFlow and want to implement **Differential Privacy**, as it has built-in privacy tools.
- **Flower**: A flexible, framework-agnostic option. It lets you use any machine learning library (like PyTorch or TensorFlow) and is well-suited for building and managing a federated learning system with multiple clients.

Machine Learning Libraries

- **PyTorch** or **TensorFlow**: The foundational libraries for building and training your neural network model for disease prediction.
- **Pandas**: Essential for handling data. You will use it to read and process the CSV/Excel files uploaded by the hospital admins.
- **NumPy**: A core library for numerical operations, which is used by both PyTorch and TensorFlow for handling model weights and data.

Web Development

- **Backend**: **Flask** is an excellent choice. It's a lightweight Python web framework that's perfect for building the central server, handling API requests from the hospital portals, and serving the dashboard.
- **Frontend**: You can use a modern JavaScript framework like **React** or **Vue.js** for the hospital portals and the central dashboard. These frameworks make it easy to build

dynamic, interactive user interfaces. For a simpler approach, you could use plain HTML, CSS, and JavaScript.

- **Database**: A database like **SQLite** (for a simple, file-based database) or **PostgreSQL** can be used to store user credentials and track training history.

Privacy-Enhancing Libraries

- **TenSEAL** (for PyTorch): A library that provides a high-level API for Homomorphic Encryption, making it easier to integrate into your PyTorch models.
- **Opacus** (for PyTorch) or **TensorFlow Privacy** (for TensorFlow): Libraries that simplify the implementation of **Differential Privacy** by automatically handling gradient clipping and noise injection.

Tools and Functions

1. Central Server

The central server is the "brain" of the operation, responsible for coordination and aggregation.

- **Tools**: **Flask** (Backend Framework), **PostgreSQL** (Database), **TensorFlow Federated (TFF)** or **PySyft** (FL Framework), **TensorFlow/PyTorch** (ML Library), **Chart.js** (Visualization).
- **Key Functions**:
  - **Dashboard (/dashboard)**: Displays real-time training metrics (accuracy, loss) and client status.
  - **API Endpoints**:
    - GET /get_global_model: Sends the current version of the global model to clients.
    - POST /aggregate_updates: Receives and aggregates the privacy-enhanced updates from all clients.
  - **Model Aggregation**: A function that combines the received model updates into a new global model.
  - **Database Management**: Handles user authentication and stores training logs.

2. Hospital Clients

Each hospital client is an independent node with its own data and user portals. This is where the local training and predictions occur.

- **Tools**: **Flask** (Local Backend), **Pandas** (Data Handling), **TensorFlow/PyTorch** (ML Library), **Opacus** (for DP) or **TenSEAL** (for HE).

- **Key Functions**:
    - **Admin Portal (/admin_login)**:
        - **Dataset Upload**: An interface to upload CSV/Excel files.
        - **Training Trigger**: A function to start local model training on the uploaded data.
        - **Status Display**: Shows local training progress, accuracy, and loss.
    - **Doctor's Portal (/doctor_login)**:
        - **Data Input Form**: A form for doctors to enter a patient's data.
        - **Prediction Function**: Loads the latest global model and makes a prediction on the single patient's data.
    - **Local Training Module**: A Python script that:
        - Loads the local model and data.
        - Performs local training.
        - Applies a privacy mechanism (**DP** or **HE**).
        - Sends the enhanced model updates to the central server's API.

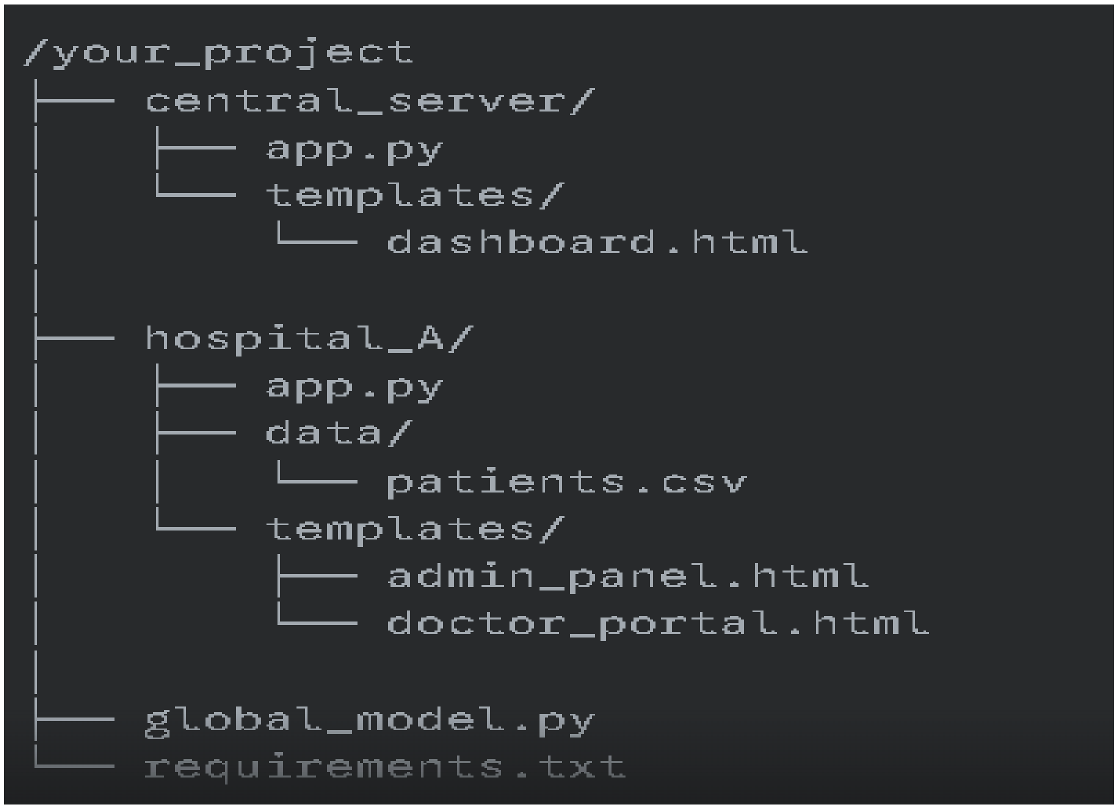
3. Data and Communication Flow

This describes the flow of information between the components.

- **Model Broadcast (Round 1)**: The **Central Server** sends the initial **Global Model** to all three **Hospital Clients**.
- **Local Training & Privacy**: Each **Hospital Client** trains the model on its **Local Dataset** and applies a privacy technique to the model updates. The raw data never leaves the hospital.

- **Update Transmission**: The clients send their **Privacy-Enhanced Model Updates** (encrypted or noisy weights) back to the **Central Server** via a secure API.
- **Aggregation**: The **Central Server** aggregates the updates to create a new, improved **Global Model**.
- **Prediction**: The **Doctor's Portal** uses the latest **Global Model** to make predictions on **Single Patient Data**. This data also remains at the hospital.

Approach 1: PySyft & Homomorphic Encryption 🔐



```
/your_project
├── central_server/
│   ├── app.py
│   └── templates/
│       └── dashboard.html
│
├── hospital_A/
│   ├── app.py
│   ├── data/
│   │   └── patients.csv
│   └── templates/
│       ├── admin_panel.html
│       └── doctor_portal.html
│
├── global_model.py
└── requirements.txt
```

This approach is ideal for a project that prioritizes strong, cryptographic privacy guarantees. PySyft is built for secure, multi-party computation, making it an excellent choice for implementing Homomorphic Encryption (HE).

Tools

- **Federated Learning Framework**: **PySyft** (built on PyTorch). It provides high-level abstractions for secure computation and federated learning.
- **Machine Learning Library**: **PyTorch**. The deep learning model will be built using this.
- **Web Framework**: **Flask**. For building the central server and the hospital portals.
- **Encryption**: The tenseal library integrates well with PyTorch for HE, allowing computations on encrypted tensors.
- **Data Handling**: **Pandas** for processing the CSV/Excel files.

Project plan

Workflow

1. **Central Server Initialization**:
    - The Flask server is set up and generates a public/private key pair for the Homomorphic Encryption scheme.
    - It initializes the PyTorch-based global model.
    - The public key and the model are sent to the three hospital clients via a secure API endpoint.
2. **Hospital Client Operations**:
    - **Admin Portal**: An admin logs in and uploads the hospital's private dataset (CSV/Excel). The data is loaded into a Pandas DataFrame.
    - **Local Training**: A Python script uses PyTorch to train the model locally on the dataset.
    - **Encryption**: After training, the model's updated parameters are encrypted using the public key with tenseal. The raw data remains unencrypted and on-site.
    - **API Call**: The encrypted model updates are sent to the central server via an API call.
3. **Central Server Aggregation**:
    - The server receives the encrypted updates from all three hospitals.
    - It uses Homomorphic Encryption to **aggregate the encrypted updates** directly, without ever decrypting them. This ensures the server cannot see the individual contributions.
    - The server decrypts the final, aggregated model and updates the global model for the next round.
4. **Prediction Portal**:

- The doctor's portal uses the latest global model (sent back from the server) to make predictions on new patient data entered via a form. This prediction occurs locally at the hospital.

Approach 2: TensorFlow Federated & Differential Privacy 🛡️

This approach is excellent for projects that need to be highly scalable and robust against data reconstruction attacks. Differential Privacy (DP) offers a strong, mathematical guarantee of privacy.

Tools

- **Federated Learning Framework**: **TensorFlow Federated (TFF)**. This is a robust framework from Google designed for large-scale federated learning.
- **Machine Learning Library**: **TensorFlow**. The model will be built with TensorFlow or Keras.
- **Web Framework**: **Flask**. For the central and client web applications.
- **Privacy**: **TensorFlow Privacy**. This library is designed to seamlessly add differential privacy to your TensorFlow models.
- **Data Handling**: **Pandas**.

Workflow

1. **Central Server Initialization**:
   - A Flask server is set up with TFF to handle the federated learning protocol.
   - The TensorFlow/Keras-based global model is initialized on the server.
   - The server broadcasts the initial model to the three hospital clients.
2. **Hospital Client Operations**:
   - **Admin Portal**: An admin uploads the hospital's private dataset.
   - **Local Training**: A script uses TensorFlow to train the model locally.
   - **Privacy Injection**: **TensorFlow Privacy** is used to add a controlled amount of noise to the model updates (gradients) during the training process. This is often

done using a differentially private optimizer. The raw data stays on the local server.

- **API Call**: The noisy, updated model parameters are sent to the central server.

3. **Central Server Aggregation**:
   - The central server receives the noisy updates from all hospitals.
   - It aggregates them using a method like FedAvg. The noise from all clients helps obscure any single individual's data, providing the privacy guarantee.
   - The global model is updated for the next training round.
4. **Prediction Portal**:
   - The doctor's portal makes predictions using the latest global model that has been sent back to the hospital from the server.

Procedures to Integrate Everything

Step 1: Backend Setup (Central Server & Hospital Portals)

- **Initialize Flask**: Set up a Flask application that will act as both your central server and the host for the hospital portals. You can use separate "blueprints" in Flask to modularize the code for each hospital.
- **API Endpoints**: Create the necessary API endpoints:
    - POST /upload_data: For hospital admins to upload their CSV/Excel files.
    - POST /send_model_updates: For hospitals to send their privacy-enhanced model updates.
    - GET /get_global_model: For hospitals to download the latest global model.
    - GET /dashboard_data: To provide real-time data for the central dashboard.
- **Database Integration**: Set up your database to store user information (for admin and doctor logins) and potentially a log of training rounds.
-

Step 2: Frontend Development (Hospital Portals & Dashboard)

- **Admin Panel**: Build a web page with a file upload form and a "Start Training" button. Use JavaScript to make an API call to the Flask backend to start the process and periodically fetch status updates.
- **Doctor's Portal**: Create a form with input fields for patient data. The form submission will trigger a local script to load the latest global model and make a prediction.
- **Central Dashboard**: Design a dashboard that fetches data from your backend API and visualizes the training progress using a library like **Chart.js** or **D3.js**.

Step 3: Federated Learning Logic

- **Client-Side (Hospitals)**:
    1. **Local Training Script**: Write a Python script that uses PyTorch or TensorFlow to train the model on the uploaded dataset.
    2. **Privacy Implementation**:
        - **HE**: Use a library like TenSEAL to encrypt the model weights after local training.
        - **DP**: Use Opacus or TensorFlow Privacy to apply a differentially private optimizer during training.
    3. **Communication**: The script will use the requests library to make POST requests to the central server's API endpoints.
- **Server-Side (Central Server)**:
    1. **Model Aggregation**: Implement a function that receives the privacy-enhanced model updates.
    2. **HE**: If using HE, this function will use the HE library to perform computations on the encrypted updates before decrypting the final aggregated result.
    3. **DP**: If using DP, the aggregation method (e.g., Federated Averaging) will be applied to the noisy updates.
    4. **Global Model Update**: The aggregated result is used to update the central server's copy of the global model.