Federated Learning for Privacy-Preserving Heart Disease Prediction

Alen Scaria

Department of Computer Science and Engineering
National Institute of Technology Rourkela

Federated Learning for Privacy-Preserving Heart Disease Prediction

A report submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Alen Scaria

(Roll Number: 121CS0237)

based on research carried out

under the supervision of

Prof. Judhistir Mahapatro

May, 2025

Department of Computer Science and Engineering

National Institute of Technology Rourkela

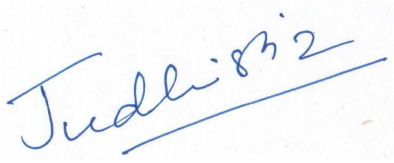
Department of Computer Science and Engineering
National Institute of Technology Rourkela

Prof. Judhistir Mahapatro

May 13, 2025

Supervisor's Certificate

This is to certify that the work presented in the project report entitled *Federated Learning for Privacy-Preserving Heart Disease Prediction* submitted by *Alen Scaria*, Roll Number 121CS0237, is a record of original research carried out by him under my supervision and guidance in partial fulfillment of the requirements of the degree of *Bachelor of Technology* in *Computer Science and Engineering*. Neither this project report nor any part of it has been submitted earlier for any degree or diploma to any institute or university in India or abroad.



Judhistir Mahapatro

Declaration of Originality

I, *Alen Scaria*, Roll Number *121CS0237* hereby declare that this project report entitled *Federated Learning for Privacy-Preserving Heart Disease Prediction* presents my original work carried out as a undergraduate student of NIT Rourkela and, to the best of my knowledge, contains no material previously published or written by another person, nor any material presented by me for the award of any degree or diploma of NIT Rourkela or any other institution. Any contribution made to this research by others, with whom I have worked at NIT Rourkela or elsewhere, is explicitly acknowledged in the dissertation. Works of other authors cited in this dissertation have been duly acknowledged under the sections "Reference" or "Bibliography". I have also submitted my original research records to the scrutiny committee for evaluation of my dissertation.

I am fully aware that in case of any non-compliance detected in future, the Senate of NIT Rourkela may withdraw the degree awarded to me on the basis of the present dissertation.

May 13, 2025
NIT Rourkela

Alen Scaria

Acknowledgment

I would like to express my sincere gratitude to Professor Judhistir Mahapatro for his valuable guidance, consistent support, and insightful suggestions throughout the course of this project. His technical expertise and timely feedback played a crucial role in shaping this work.

I would also like to extend my heartfelt thanks to my colleague Amal Mohan for his unwavering support and collaboration during the development and execution of this project.

My appreciation goes to all my classmates, whose ideas, feedback, and encouragement contributed meaningfully to the progress of this research.

Finally, I am grateful to the Department of Computer Science and Engineering, NIT Rourkela, for providing the academic resources and a conducive environment for research and learning.

May 13, 2025
NIT Rourkela

Alen Scaria
Roll Number: 121CS0237

Abstract

Heart disease remains one of the leading causes of mortality worldwide, necessitating early and accurate detection methods. Traditional healthcare systems often rely on centralized data processing, which raises significant concerns around data privacy, security, and communication overhead, especially in resource constrained or remote environments. This thesis presents a decentralized approach to heart disease prediction using **Federated Learning (FL)**, integrated with IoT-inspired health monitoring systems. The proposed framework simulates edge-based deployment using devices like Raspberry Pi, where sensor data is processed locally, and only model parameters are shared with a central server for aggregation. This methodology ensures data remains private while enabling collaborative model training across multiple clients. An Artificial Neural Network (ANN) was employed as the predictive model, optimized for low-complexity execution on constrained devices. The system demonstrates the feasibility of using FL for privacy-preserving and scalable heart disease prediction. It also lays the groundwork for future enhancements, such as real-time sensor integration and more complex deep learning architectures tailored for edge environments.

Keywords— *Federated Learning; Heart Disease Prediction; IoT Healthcare; Privacy-Preserving AI; Edge Computing; Artificial Neural Network (ANN); Raspberry Pi; Decentralized Machine Learning.*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The healthcare industry is undergoing a rapid transformation through the adoption of Internet of Things (IoT) and Artificial Intelligence (AI) technologies. IoT-enabled devices and wearables are increasingly used for remote patient monitoring, especially for individuals with cardiac conditions, offering early detection and timely medical interventions [2].

Conventional centralized data processing systems present notable limitations, including concerns over data privacy, significant communication demands, and challenges in detecting anomalies in real-time. With ongoing advancements in remote monitoring technologies, decentralized approaches are becoming more practical, allowing vital signs to be tracked without requiring physical presence at healthcare facilities.

1.1 Problem Definition

Legacy heart monitoring systems also encounter several obstacles. Their dependence on centralized architectures heightens privacy risks, increases communication loads, and makes timely anomaly detection difficult. Monitoring through a single device often yields limited insights due to restricted access to comprehensive local data. Furthermore, real-time data transmission is hindered in regions with low bandwidth availability. While remote monitoring holds promise, its success hinges on the development of efficient and secure mechanisms to overcome these technical barriers.

1.2 Objectives

This project introduces a decentralized approach to heart disease prediction by implementing Federated Learning (FL) across several Raspberry Pi devices operating as edge nodes. The primary focus is to protect patient confidentiality while enabling local training of machine learning models on individual devices. By avoiding centralized data aggregation, the system reduces communication costs and strengthens data privacy. Additionally, the

project evaluates the system's performance in terms of predictive accuracy, consistency, and operational efficiency under practical conditions.

1.3 Literature Review

Federated Learning (FL) has gained significant momentum in the healthcare and IoT fields due to its ability to support privacy-preserving machine learning with lower communication overhead. Numerous studies have explored how FL can be employed to securely process sensitive medical information directly at the edge, maintaining data confidentiality.

Rudraraju et al. [3] explored the use of FL in a fog computing-based smart home system, where heterogeneous sensor data from IoT devices were processed locally at edge nodes. These local models were then aggregated at a central node, avoiding the need to transmit raw data and thus maintaining privacy. This setup proved particularly effective in bandwidth-limited environments, presenting a compelling use case for real-time remote healthcare monitoring.

Focusing specifically on heart disease prediction, Kavitha. B. S et al. [4] applied FL to the Cleveland dataset using traditional machine learning models such as Logistic Regression and Support Vector Machines (SVM). Their work demonstrated the practicality of FL for healthcare by achieving meaningful predictions without centralized data storage.

Bebortta et al. [5] proposed a federated approach named FedEHR, utilizing a soft-margin L1-regularized SVM (sSVM) on electronic health records (EHR). Their design emphasized privacy-preserving training across decentralized platforms, further underscoring FL's role in sensitive health data processing.

Ancy J. J. et al. [6] adopted FL with data mining and classical ML techniques such as K-Nearest Neighbors (KNN), Decision Tree (DT), and Naive Bayes (NB) to classify heart disease severity. Using the Cleveland dataset, they highlighted the feasibility of employing lightweight models at the edge using FL.

Gupta et al. [7] explored a deep learning-based approach with multiple dense layers in a federated setting. Although details regarding the dataset and training procedures were limited, their comparative analysis with conventional ML models reflected the growing interest in neural networks for FL in healthcare.

Regarding architectural choices, Sonawane and Patil [8] evaluated the use of multilayer perceptron (MLP) neural networks on the Cleveland dataset. By varying the number of neurons and features, they assessed prediction accuracy and demonstrated the applicability of ANNs in heart disease classification.

Motivated by these findings, our work adopts a lightweight Artificial Neural Network (ANN) with a reduced number of input features and neurons. This design choice accommodates the limited computational resources available on devices like the Raspberry Pi. Although the simplified architecture may affect overall accuracy, in medical applications,

even less precise predictions—such as false positives—can be valuable, prompting timely interventions and preventive healthcare responses.

Chapter 2

Methodology

This chapter presents a clear overview of the methodology employed in this research. This chapter explains the methods, the tools, and the research frameworks involved, offering some insight into the way in which each of the methods lends support to the accomplishment of the project objectives.

2.1 Artificial Neural Network

Artificial Neural Networks (ANNs) are the pillars of artificial intelligence and machine learning. Following the pattern of the human brain, ANNs are used extensively to solve complex problems across various industries, from natural language processing, image and speech recognition, to decision-making.

Essentially, ANNs comprise interconnected nodes (or neurons) organized into layers: input layers, hidden layers, and output layers. The nodes perform mathematical computations such as weighted summations and apply activation functions, enabling the network to represent complex relationships within data.

ANNs are typically trained using **supervised learning**, where the model learns to map inputs to outputs by minimizing prediction errors. This training is conducted by adjusting internal parameters (weights and biases) using optimization techniques like **gradient descent** and **backpropagation**.

A major advantage of ANNs is their ability to automatically learn useful features from raw input data through hierarchical layer processing. This makes them highly effective in handling high-dimensional and complex datasets.

Over time, specialized neural network architectures have emerged to address specific tasks:

- **Convolutional Neural Networks (CNNs)** – image processing,
- **Recurrent Neural Networks (RNNs)** – sequential data,

- **Transformers** – natural language understanding.

Despite challenges such as large data requirements, complex architectures, and limited interpretability, ANNs remain a central part of modern AI, powering a vast range of innovative applications.

2.1.1 Architecture

Artificial Neural Networks (ANNs) consist of connected processing units called neurons (or nodes), arranged in layers. The structure is inspired by biological neurons in the human brain, where signals are transmitted and processed through weighted connections.

ANNs are widely used in areas such as healthcare, where they assist in diagnosis, patient monitoring, and predictive analytics.

Layers in an ANN

An ANN generally has three types of layers:

A. Input Layer

The input layer is the initial stage of an artificial neural network where raw data enters the model. Each neuron in this layer corresponds to a single input feature from the dataset, such as a pixel in an image or a numerical value in a dataset. Unlike other layers, the input layer does not perform any computations; its sole function is to pass the received data to the subsequent hidden layer for further processing. For instance, in an image recognition task, each neuron in the input layer might represent the intensity value of a pixel.

B. Hidden Layer(s)

The hidden layers are the core computational components of a neural network. Each neuron of the hidden layer takes inputs from the previous layer, applies a set of weights and a bias, and processes the result using an activation function. Deep neural networks may contain multiple hidden layers, which allow the model to learn complex, hierarchical representations of the data. For example, in image processing, the first hidden layer might detect simple patterns such as edges, while deeper layers build on these to identify more abstract features like shapes or objects. The architecture, including the number of hidden layers and the number of neurons in each, is a hyperparameter that can be tuned based on the task.

C. Output Layer

The output layer generates the final predictions of the neural network. The number of neurons in this layer depends on the specific task. For regression tasks, there is typically one neuron to output a continuous value. In binary classification tasks, one neuron is used with a sigmoid activation function to output a value between 0 and 1, indicating the probability of one class. For multi-class classification, the output layer contains multiple neurons, with

each neuron representing a class and using a softmax activation function to produce the probability distribution across all classes.

Neurons and Their Components

Each neuron processes input data in the following steps:

A. Weighted Sum of Inputs

Each input x_i is multiplied by a corresponding weight w_i, and a bias term b is added. The total input to the neuron is the weighted sum of the inputs plus the bias. This is represented mathematically as:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

This total input z is then passed to the activation function, which determines the output of the neuron.

B. Activation Function

This function introduces non-linearity to help the network learn complex patterns.

- Common activation functions:
  - Sigmoid: Outputs values between 0 and 1.
  - ReLU (Rectified Linear Unit): Outputs x if $x > 0$, else 0.
  - Softmax: Converts outputs into probabilities (used in multi-class classification).

C. Output Propagation

The output after activation, $a = f(z)$, is passed to the next layer.

Weights and Biases in ANNs

In an Artificial Neural Network (ANN), every connection between neurons has a weight associated with it, which controls the strength and impact of one neuron's output on another. The weights are important as they determine the degree to which an input signal is added to the activation of the neuron. The weights are generally initialized before training using techniques such as He or Xavier initialization so that learning becomes stable and efficient. In addition to weights, there is also a bias term in each neuron, which is an adjustable offset and enables the activation function to move left or right for improved fitting of the data. In training, both biases and weights are iteratively updated using backpropagation, in which the network reduces error by calculating gradients and using optimization methods such as gradient descent or its variations (e.g., Adam, RMSprop). This ongoing refinement allows the ANN to learn intricate patterns and enhance prediction accuracy in the long run.

2.1.2 Learning Process in Artificial Neural Networks (ANNs)

The process of learning in ANNs is an iterative process of adjusting weights and biases to reduce prediction errors. This is done by combining forward propagation, computation of loss, backpropagation, and optimization. Normalization methods are also commonly used to provide stable and efficient training.

Forward Propagation

Input data is passed through the network layer by layer. Each neuron computes a weighted sum of its inputs, adds a bias, and applies an activation function:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

where w_i are the weights, x_i are the inputs, and b is the bias.

The activation function is then applied to the weighted sum to get the output of the neuron:

$$a = f(z)$$

where $f(z)$ is the activation function, and a is the output of the neuron.

The final output (prediction) is compared to the true value using a loss function (e.g., Mean Squared Error for regression, Cross-Entropy for classification).

Loss Function

The loss function quantifies how far the predicted output is from the actual target.

Common loss functions include:

Mean Squared Error (MSE) for Regression:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

where y_i is the true value, $\hat{y}_i$ is the predicted value, and N is the number of data points.

Cross-Entropy for Classification:

$$\text{Cross-Entropy} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

where y_i is the true label (one-hot encoded), and $\hat{y}_i$ is the predicted probability for class i.

Backpropagation

Backpropagation is the neural network's way of efficiently calculating how each weight contributes to the overall error. After the forward pass computes predictions and loss, backpropagation works backward through the network, applying the chain rule to determine the exact gradient (partial derivative) of the loss with respect to each weight. These gradients reveal both the direction and magnitude by which each weight should be adjusted to reduce error.

Gradient descent then puts this information to work. Using the gradients computed during backpropagation, it updates each weight by taking a small step (controlled by the learning rate) in the direction that minimizes the loss. This iterative process—calculating gradients via backpropagation and adjusting weights via gradient descent—is how neural networks gradually improve their performance, fine-tuning their parameters to make better predictions over time. Together, they form the core optimization engine that drives learning in neural networks.

Training and Optimization

The weight update process is where learning occurs in neural networks. After backpropagation computes the gradients, optimization algorithms adjust the weights and biases. While basic gradient descent subtracts a fraction (learning rate) of the gradient, advanced optimizers like Adam and RMSprop improve efficiency by using momentum and adaptive learning rates. These methods accelerate learning in consistent directions and reduce oscillations. Choosing the right learning rate is crucial—too high causes instability, while too low slows progress. Regularization techniques like weight decay are also used to prevent overfitting by penalizing large weights. This update cycle repeats over many iterations to gradually minimize the loss function.

Normalization

Normalization techniques are essential for stable and efficient neural network training. The process begins by standardizing input features to keep them on similar scales, preventing any one feature from dominating. Within networks, batch normalization standardizes layer outputs per mini-batch, reducing internal covariate shift and enabling higher learning rates. It also includes learnable parameters to retain the model's flexibility. For models like RNNs and Transformers, layer normalization is used instead, normalizing across features rather than batches. These methods speed up convergence, improve generalization, and reduce sensitivity to initial conditions, making them vital for training deep learning models effectively.

2.1.3 Adaptation of ANN in project

Artificial Neural Network (ANN) is utilized as the core classification model for heart anomaly detection within our federated learning framework. Due to the resource limitations of the Raspberry Pi 3 Model B, a lightweight ANN architecture was implemented to ensure efficient execution on edge devices.

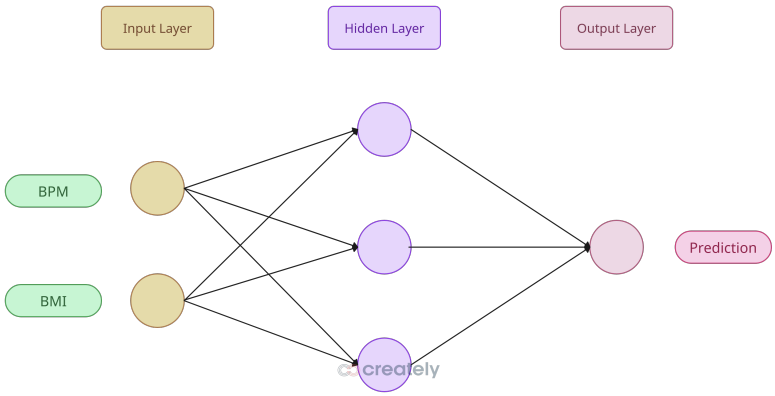


Figure 2.1: ANN Model

The ANN model is structured as follows:

- 2 input features: Body Mass Index (BMI) and Beats Per Minute (BPM)
- 1 hidden layer with 3 neurons
- 1 output neuron that predicts the presence or absence of heart disease (binary classification)

The sigmoid activation function is applied throughout the network to facilitate binary decision-making. To optimize the model for Raspberry Pi's hardware constraints, the ANN is implemented using NumPy, avoiding high-overhead libraries like TensorFlow or PyTorch.

In our federated learning workflow, each Raspberry Pi client trains the ANN locally using real-time, device-specific data. Only the updated model weights are transmitted to a central server, ensuring data privacy. The server aggregates these weights and sends the updated model back to the clients. This federated training loop is performed over three rounds, allowing all devices to collaboratively learn without sharing raw data.

2.2 Federated Learning

Traditional machine learning models often rely on a centralized approach, where data from various sources is aggregated and stored on a central server for training. While this method

can be effective, it poses significant challenges, especially concerning data privacy and security. Centralizing sensitive information, such as medical records, increases the risk of data breaches and unauthorized access. Additionally, transferring large volumes of data to a central location can lead to substantial communication overhead and may not be feasible in environments with limited bandwidth or strict data governance regulations.

Federated Learning (FL) emerges as a solution to these challenges by enabling decentralized model training. In FL, individual devices or clients retain their local data and train models independently. Instead of sharing raw data, clients send only the learned model parameters (e.g., gradients or weights) to a central server.[3] The server then aggregates these parameters to update the global model. This approach enhances data privacy, reduces the risk of data breaches, and minimizes communication overhead, making it particularly suitable for applications in healthcare and other sensitive domains.

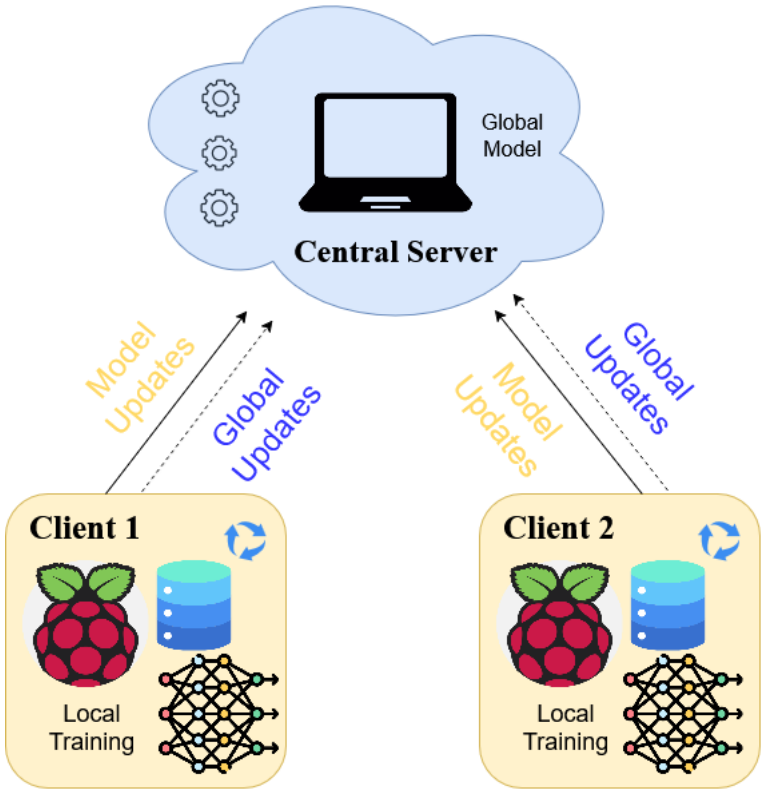


Figure 2.2: Federated Learning Architecture

2.2.1 How Federated Learning works

The process of Federated Learning (FL) involves several key steps that collectively enable decentralized model training while preserving data privacy:

- **Model Initialization:** A global model is initialized on a central server.

- **Model Distribution:** The initialized model is distributed to multiple client devices (e.g., smartphones, IoT devices).
- **Local Training:** Each client trains the model locally using its own data, ensuring that raw data remains on the device.
- **Update Sharing:** After local training, clients send only the updated model parameters (e.g., gradients or weights) to the central server.
- **Aggregation:** The server aggregates the received updates, typically using algorithms such as Federated Averaging (FedAvg), to update the global model.
- **Model Redistribution:** The updated global model is redistributed to the clients for the next round of training.
- **Iteration:** Steps 3–6 are repeated over multiple rounds until the model achieves the desired level of performance.
- **Data Privacy Preservation:** At no point does raw data leave the client devices, thereby maintaining data privacy and security throughout the process.

This iterative process enables the development of accurate and robust machine learning models while significantly reducing communication overhead and safeguarding sensitive information.

2.3 FedAvg

Commonly used aggregation methods in Federated Learning include Federated Averaging (FedAvg), Federated Stochastic Gradient Descent (FedSGD), and others. Among these, the most widely adopted is FedAvg, which is also the method used in this thesis. The fundamental idea behind FedAvg is that N clients begin with a shared initial model w, train it locally on their individual datasets, and then send the updated model parameters back to the central server. Once the server receives the N locally trained models, it computes a weighted average of these parameters to update the global model.

A key advantage of FedAvg over FedSGD is that clients can perform multiple local training epochs before communicating with the server. Unlike FedSGD—which transmits gradients after every mini-batch—FedAvg sends only the final model parameters after local training. This significantly reduces communication overhead.

We are considering a server node where the model will be trained initially and client nodes will clone this server model and where this model is trained and tested using client's

local data and then just the updated weights are passed on to the Server where the weights are aggregated to update the individual weights. In summary, FedAvg is considered more

Algorithm 1 Federated Learning

Server-Side:
Initialize global model weight w_0
for each round $t = 0, 1, \ldots$ **do**
  for each client $i = 0, \ldots, n-1$ **in parallel do**
    Send w_t to client i
    Receive updated weight $w_{t+1}^{(i)}$ from client i
  end for

$$w_{t+1} \leftarrow \frac{1}{n} \sum_{i=1}^{n} w_{t+1}^{(i)}$$

end for

Client-Side:
Split local dataset into batches of size B
for each local epoch $e = 0, \ldots, E-1$ **do**
  for each mini-batch b of size B **do**
    Update local model:

$$w_t = w_t - \eta \cdot \nabla L(w, b)$$

  end for
end for
Return updated weight w_{t+1} to the server

Figure 2.3: Federated Averaging algorithm. [1].

efficient and scalable than FedSGD, especially in scenarios with limited bandwidth and high communication costs, due to its reduced transmission frequency and enhanced system efficiency.

2.4 Flower Framework

Flower[1] is a flexible, open-source platform built to facilitate federated learning across multiple devices or systems. Developed in Python, it offers a convenient way to train various machine learning models, including deep learning architectures. One of its standout features is an intuitive high-level API that hides much of the complexity involved in tasks like splitting datasets, distributing models, handling communication between nodes, and performing model aggregation. This makes it easier for developers and researchers to set up and manage federated learning experiments without needing to dive into the lower-level implementation details.

Flower supports various federated optimization algorithms, including Federated Averaging

[1]Source:`https://flower.ai/docs/`

(FedAvg), FedProx, and Federated SGD. It also provides the flexibility to define and implement new optimization strategies with ease. The framework operates on a client-server architecture, requiring at least one server and a minimum of two clients to initiate federated training.

Key Features of Flower:

- **Client-Server Architecture:** Flower uses a client-server model where clients train models locally and send updates to a central server for aggregation, facilitating decentralized training while preserving data privacy.
- **Flexible Client Management:** Flower supports dynamic client management, accommodating clients with varying computational capabilities and network conditions. It allows for both synchronous and asynchronous update mechanisms.
- **Custom Aggregation Strategies:** Flower enables the use of different aggregation methods such as FedAvg and provides the flexibility to implement custom strategies for combining local models into a global model.
- **Scalability:** The framework is designed to scale from a small number of clients to thousands, making it suitable for both small-scale experiments and large-scale federated learning deployments.
- **Cross-Platform Support:** Flower is compatible with a wide range of platforms, from edge devices like Raspberry Pi to cloud-based environments.
- **Privacy-Preserving:** By ensuring that data remains on local devices throughout the training process, Flower significantly reduces the risk of data breaches, making it well-suited for sensitive domains such as healthcare and finance.

The simplicity, flexibility, and scalability of Flower make it an ideal framework for implementing federated learning, effectively bridging the gap between academic research and real-world applications.

2.5 Dataset Overview

For our system for detecting heart abnormalities, we used the standard Heart Disease Dataset from Kaggle, with 303 patient records and 14 clinical features. For compatibility with our resource-limited Raspberry Pi 3 Model B edge devices, we judiciously concentrated on two clinically relevant features: Body Mass Index (BMI), computed from patient weight and height to determine cardiovascular risk factors, and Beats Per Minute (BPM), computed from maximum heart rate readings to determine cardiac function. The dataset delivers binary

labels (0 for normal, 1 for abnormal) reporting heart abnormality existence. This sparse but semantically significant feature selection is clinically relevant while also facilitating efficient implementation of federated learning, under which models locally train on edge devices and contribute only parameter updates—never unparsed patient data—to the central server, thus ensuring privacy while providing impactful cardiovascular risk estimation capabilities.[2]

[2]Source:`https://www.kaggle.com/datasets/shaukathussain/augmented-health-heart-rate`

Chapter 3

Hardware Overview

Our project utilizes a combination of IoT hardware components to collect real-time data and implement federated learning for anomaly detection. Below is a detailed description of each component.

3.1 Raspberry Pi 3 Model B (Client)

The Raspberry Pi 3 Model B is a power-efficient, miniaturized single-board computer powered by a quad-core ARM Cortex-A53 1.2 GHz processor with 1GB of RAM. It features Wi-Fi, Bluetooth, Ethernet, USB connectivity, and 40 GPIO pins, making it an excellent choice for IoT and edge computing applications that require connectivity, minimal power usage, and flexibility. Its small form factor and low power consumption enable deployment in resource-limited environments, including remote locations.

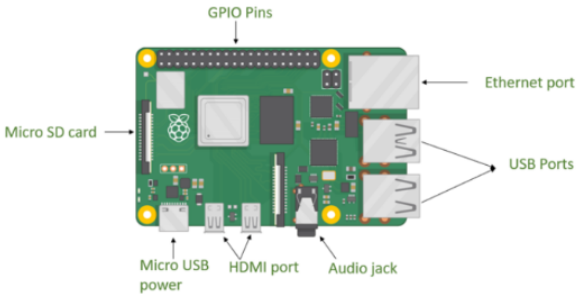


Figure 3.1: Raspberry Pi 3

In this project, the Raspberry Pi 3 Model B functions as an edge device that collects and processes diabetes-related information, such as BMI, blood pressure, and age, received from Arduino sensors. The device stores the data locally and participates in training a federated learning model to predict the risk of diabetes. Notably, Raspberry Pi ensures data privacy by not transmitting raw data but instead sending only the model updates to the central server. This approach reduces the risk of privacy breaches and minimizes

communication overhead, which is particularly valuable in settings with limited network bandwidth.

Raspberry Pi is well-suited for local machine learning tasks, where it initially performs computations and model updates before transferring the results to a central server. This minimizes communication overhead while ensuring secure and efficient data processing. Moreover, the Raspberry Pi's versatile GPIO pins allow seamless connection with external sensors, enabling the collection of real-time data. Its ability to support multiple communication protocols, including Wi-Fi and Ethernet, positions it as an ideal platform for deploying a decentralized, privacy-preserving federated learning network for diabetes prediction.

Key Applications of Raspberry Pi:

- **IoT and Smart Home Automation:** Used to control smart devices, monitor environmental conditions, and automate tasks in homes or businesses.
- **Edge Computing and Data Processing:** Raspberry Pi can handle local data processing and federated learning tasks, reducing reliance on centralized cloud computation.
- **Robotics:** Serves as the control unit for robots, enabling tasks such as navigation, object detection, and autonomous actions.
- **Media Centers and Gaming:** Popular for creating home theater systems, streaming media, and emulating retro gaming consoles.
- **Education and DIY Projects:** Ideal for teaching programming, electronics, and hands-on STEM projects, making it an excellent tool for students and hobbyists.

3.2 Arduino Uno

The Arduino Uno is an open-source microcontroller board based on the ATmega328P chip. It is one of the simplest and most universal microcontroller boards, widely used for electronics projects, especially in embedded systems and IoT applications. In this project, the Arduino Uno serves as the data acquisition unit, interfacing with sensors like the blood pressure sensor to collect real-time physiological data. This data is then transmitted to the Raspberry Pi for further processing and federated model training.

Figure 3.2: Ardinuo Uno

Key Components of Arduino Uno:

- **ATmega328P Microcontroller:** The core of the board, responsible for executing code and controlling connected peripherals.
- **Digital I/O Pins (14 total):** Used to connect digital sensors or devices; 6 of these pins support PWM (Pulse Width Modulation) functionality.
- **Analog Input Pins (6 total):** These pins are used to read analog signals from sensors, such as pulse or pressure sensors.
- **USB Port:** Facilitates programming and serial communication with a computer or Raspberry Pi.
- **Power Supply:** The Arduino can be powered either through a USB connection or via an external power jack (7-12V recommended).
- **Voltage Regulator:** Converts and stabilizes the input voltage to 5V or 3.3V for the board and peripherals.
- **Reset Button:** Allows for resetting the microcontroller and restarting the program.
- **Crystal Oscillator:** Provides accurate timing for internal processes (typically 16 MHz).

How It Works:

- **Sensor Connection:** Sensors are connected to the Arduino's analog or digital input pins. For example, a pulse sensor is connected to an analog pin to read heart rate signals.
- **Data Acquisition:** The microcontroller collects analog or digital signals from the sensors, which correspond to vital signs such as BPM or blood pressure.
- **Signal Processing (Optional):** Basic data processing, such as filtering or thresholding, can be performed on the Arduino to clean up the sensor data before transmission.

- **Serial Communication:** The Arduino transmits the collected data to the Raspberry Pi via UART (Universal Asynchronous Receiver/Transmitter) through USB or GPIO serial pins.
- **Loop Execution:** The program, written in the Arduino IDE, runs continuously, enabling real-time monitoring and data capture.

The Arduino Uno plays an importatnt role in the edge layer of the system, acting as the interface between medical sensors and the computing unit (Raspberry Pi). Its simplicity, affordability, and extensive library support make it a powerful tool for rapidly developing health monitoring solutions.

3.3 Pulse Sensor

The Pulse Sensor is an efficient, low-power heart-rate sensor designed for easy integration with Arduino. It connects directly to the Arduino board and can be conveniently attached to a fingertip or earlobe. Compact and button-shaped, it features holes that allow it to be sewn into fabric for wearable applications.

Overview of Pulse Sensor

The sensor's front side, easily identified by a heart symbol, is where a finger should be positioned for readings. This area contains a small circular port that emits green light from an inward-facing LED. Just beneath this LED lies a light-sensitive component, the APDS-9008 ambient light sensor, commonly found in modern electronic devices like smartphones and tablets for automatic screen brightness adjustment.

On the rear side of the sensor module, you'll find essential electronic components, including the MCP6001 operational amplifier from Microchip. Accompanying these are several resistors and capacitors arranged to create a resistor-capacitor (RC) filter network. Additionally, a diode is incorporated to provide reverse polarity protection, preventing damage in case the power connections are mistakenly reversed. The device functions within a voltage range of 3.3V to 5V DC and typically draws less than 4mA of current.

The pulse sensor operates on the principle of photoplethysmography (PPG), a non-invasive optical technique for measuring heart rate. It works by projecting green light—typically around a 550nm wavelength—onto the skin and detecting the reflected light with a photosensor. Oxygen-rich hemoglobin in arterial blood absorbs green light more effectively, meaning the intensity of reflected light decreases as blood volume increases during a heartbeat. This variation is used to calculate pulse rate.

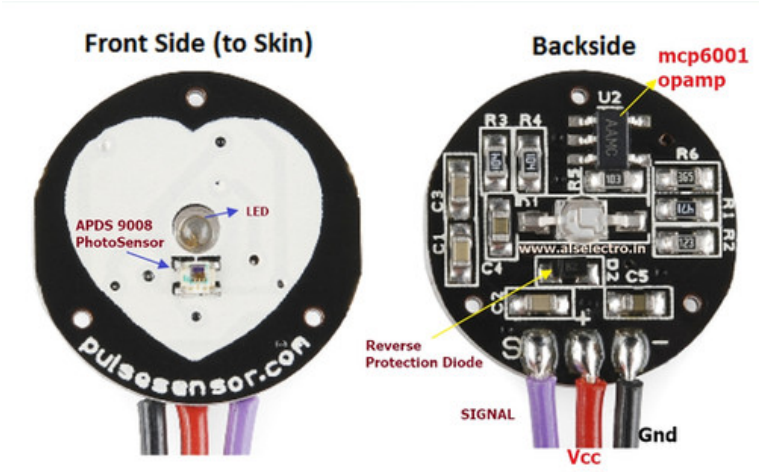


Figure 3.3: Pulse Sensor

How Pulse Sensor Works

Each time the heart beats, blood flow through the finger changes, which in turn alters the amount of light reflected back to the sensor. These variations in reflected light generate a fluctuating signal from the photosensor. By continuously illuminating the skin and capturing readings, the device can detect and track the rhythm of heartbeats over time.

Chapter 4

Implementation

To simulate the role of IoT edge devices in a federated learning environment, Raspberry Pi 3 Model B units were used due to their portability, low power consumption, and suitability for lightweight machine learning tasks. Each device acted as a client in the federated network.

4.1 Raspberry Pi OS Installation

Raspberry Pi 3 B boards were available. So based on their compatibility, a 32-bit version of Raspberry Pi OS (Lite) was chosen for better compatibility and performance. The operating system image was flashed onto a 16 GB USB drive using Raspberry Pi Imager. This external USB was used as the primary boot device to overcome SD card storage problems and to provide a slightly more stable and faster file system access.

The Raspberry Pi 3 B has only 1 GB of RAM, which is often insufficient for running multiple processes or handling memory-intensive tasks like model training. To mitigate this, the swap file size was increased manually by editing the `/etc/dphys-swapfile` configuration file. The swap size was increased. This adjustment provided better stability and allowed the training process to run with fewer crashes or slowdowns, even though with some trade-offs in speed.

4.2 RealVNC Connection using SSH

Since the Raspberry Pi units were without any dedicated monitor, mouse, or keyboard, remote access was essential for setup and ongoing interaction. SSH was enabled on each device, and RealVNC was installed to allow graphical desktop access from another machine. This setup significantly streamlined the development process, allowing code deployment, monitoring, and log analysis to be performed remotely.

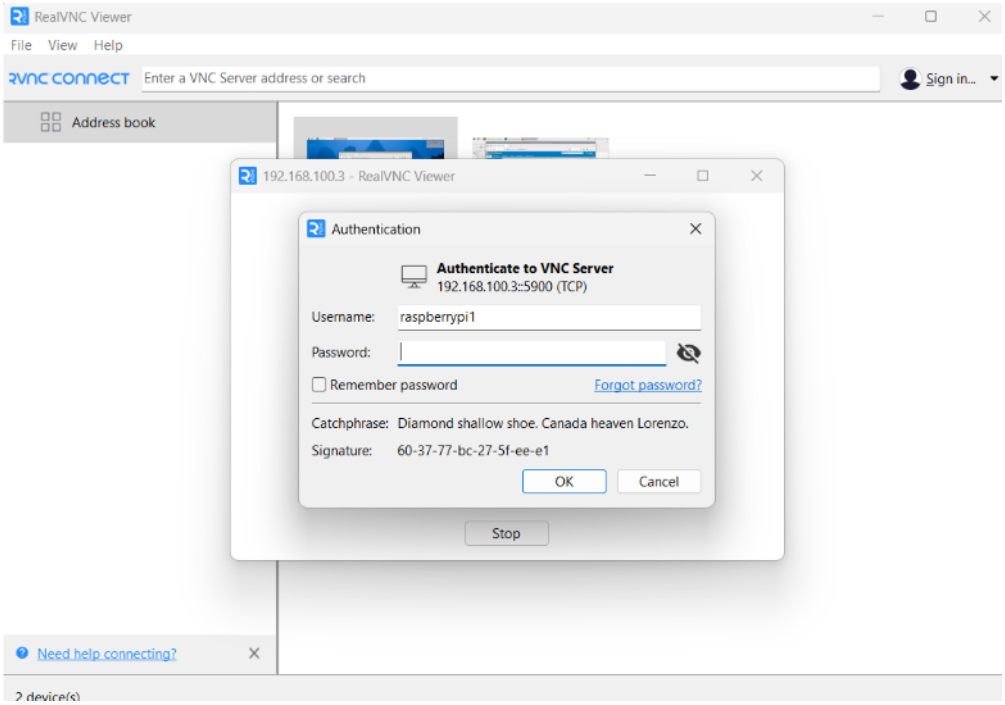


Figure 4.1: RealVNC Interface

4.3 Setup

The setup consisted of two Raspberry Pi 3 Model B devices acting as FL clients and a single laptop functioning as the central server. All three devices were connected to the same local Wi-Fi network to facilitate communication. Flower, the federated learning framework used, leverages gRPC for efficient client-server communication. This protocol allows lightweight and reliable data transfer over the network, making it suitable for resource-constrained edge devices like Raspberry Pis.

Basic connectivity was verified using `ping` commands from the server to the clients and vice versa. This ensured that all clients were reachable and ready to participate in federated rounds coordinated by the Flower server.

4.4 Dataset Split and Load

To simulate the core principle of federated learning, that data remains decentralized and local to the client devices, a pre-existing healthcare dataset was used. While real IoT sensors would generate live data on edge devices in a production environment, this project uses static datasets to emulate that scenario. Each client device is treated as if it were independently collecting health data, allowing us to test and demonstrate the FL concept.

An existing dataset from Kaggle was used. The original dataset contained multiple

health-related features. For this simulation, only the following features were retained:

- Heart Rate (BPM)
- Body Mass Index (BMI)
- Disease label (binary: 0 or 1)

Each dataset was then normalized using standard score normalization (z-score), transforming features to have zero mean and unit variance. This step improves model convergence and ensures that features with different scales do not disproportionately affect training.

4.5 Client-Side Data Distribution

Following preprocessing, the dataset was partitioned to simulate a real-world FL scenario where data is inherently distributed across clients and remains private. In this case, the dataset was divided into two equal and independent subsets (IID—independent and identically distributed) corresponding to the two Raspberry Pi clients.

Each client received a CSV file (`client_0.csv`, `client_1.csv`) containing its respective portion of the data. These files were manually transferred to the devices. Once loaded, each client locally trained its model on its private data and never shared raw data with the server. This setup closely simulates the core principle of federated learning, where the server orchestrates learning without access to user data.

4.6 Model and Flower Setup

Due to the limited computational resources and package compatibility issues on Raspberry Pi 3 B, it was not feasible to use high-level machine learning libraries like TensorFlow, PyTorch, or even TensorFlow Lite. Instead, a lightweight artificial neural network (ANN) was implemented using `NumPy`.

The ANN model was designed with a minimal structure suitable for binary classification tasks:

- **Input Layer:** 2 neurons (for 3 input features)
- **Hidden Layer:** 1 hidden layer with 3 neurons
- **Output Layer:** 1 neuron (for binary classification output)

The Sigmoid activation function was used for both the hidden and output layers, introducing non-linearity and enabling probabilistic interpretation of output values. Training

was performed using Stochastic Gradient Descent (SGD), where weights were updated after each training example within an epoch.

Hyperparameter	**Value**
Learning Rate	0.01
Number of Epochs	10
Activation Function	Sigmoid
Loss Function	Mean Squared Error (MSE)

Table 4.1: Model Training Hyperparameters

This setup ensured the model was simple enough to run on Raspberry Pi without exceeding memory or processing constraints, yet expressive enough to capture patterns in the health data.

4.7 Federated Learning Setup with Flower

The federated training was orchestrated using the Flower framework, which simplifies FL by abstracting client-server communication over gRPC. The FL setup was divided into two main components: client and server scripts.

4.7.1 Client-Side Implementation (`client.py`)

Each Raspberry Pi device ran a client script implementing a custom `NumPyClient`, where the following key methods were defined:

- `get_parameters()`
  Returns the current local model weights to the server.
- `fit(parameters, config)`
  Receives updated weights from the server, updates the local model, trains on the local dataset, and returns the new weights along with the number of examples used.
- `evaluate(parameters, config)`
  Evaluates the received model on the client's validation data and returns metrics such as loss and accuracy.

This modular structure allowed each client to operate independently, performing local computation without exposing any raw data.

4.7.2 Server-Side Implementation (`server.py`)

The server script hosted on the laptop was responsible for coordinating training rounds and aggregating client updates using the FedAvg algorithm. A custom strategy class extending

Flower's `FedAvg` was implemented to enhance visibility into the learning process. This custom class allowed for:

- Logging per-round global accuracy and loss
- Controlling the number of training rounds
- Monitoring client participation and response metrics

Since there is no centralized data for the server to compute loss, the `FedAvg` strategy computes a weighted loss using updates received from the clients.

$$\text{Global Loss} = \frac{\sum_{k=1}^{K} n_k \cdot L_k}{\sum_{k=1}^{K} n_k}$$
$$\text{where} \quad n_k = \text{number of samples in client } k$$
$$L_k = \text{local loss of client } k$$

4.8 Simulation

The simulation was designed to closely follow a real-world FL deployment, with each client independently training on its private data and communicating only with the central server.

To initiate the training process, each client was configured to connect to the IP address of the server. This IP address was manually specified in the client code to establish a gRPC connection over the local network. Flower handles the low-level communication setup internally, so clients only need to be able to reach the server's address to register for training.

Since this simulation involved only two clients, the server was set to wait until both clients were connected before beginning the first round of federated training. This synchronous setup ensures that all participating clients contribute in every round. In practical, large-scale federated learning systems, this can be extended to support partial participation, where only a subset k of n clients are required to be online for a round to begin, enabling more robust and scalable training.

4.8.1 Federated Training Rounds

Once all clients are connected, the server begins the federated training rounds. In each round:

- The server sends the current global model weights to all clients.
- Clients train the model on their local dataset for a fixed number of epochs (in this case, 10 epochs).
- Each client sends the updated weights, training loss, and accuracy back to the server.

- The server aggregates the updates using the `FedAvg` algorithm and computes the global model's evaluation metrics.

For this simulation, the system was configured to run for 3 training rounds, after which training was stopped.

```
INFO :      Starting Flower server, config: num_rounds=3, no round_timeout
INFO :      Flower ECE: gRPC server running (3 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]
INFO :      configure_fit: strategy sampled 2 clients (out of 2)
INFO :      aggregate_fit: received 2 results and 0 failures

[Round 1] Received weights from clients:
Client 0 weights:
  Layer 0 weights:
[[ -3.34147042  -3.25449577  -6.69596846]
 [  6.41441036   5.85700266 -17.32869116]]
    mean=-3.058202, std=8.022990
  Layer 1 weights:
```

Figure 4.2: FL Round Simulation

4.8.2 Logging and Evaluation

Throughout the training process, the server logs key metrics such as:

- Local accuracy and loss reported by each client
- Aggregated global accuracy and loss after each round

These metrics provide insight into how the model performance evolves over time and help identify issues like client drift or unstable convergence.

At the end of the simulation, the collected performance data was visualized using `Matplotlib`. Plots were generated to show:

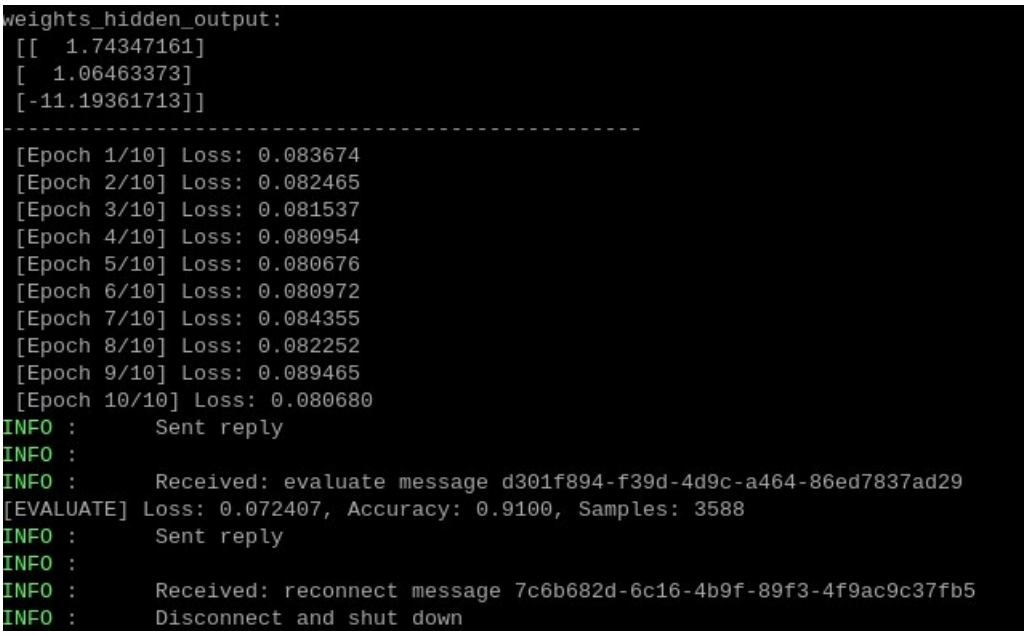


Figure 4.3: End Of Simulation

- Client-wise training and validation accuracy per round
- Global accuracy and loss trends across rounds

These visualizations aid in analyzing the effectiveness of the federated approach and validating that learning occurred consistently across the distributed setup.

Chapter 5

Results and Discussion

The performance of the federated learning setup was evaluated using metrics collected during each round of training. `Matplotlib` was used to visualize global and client-level trends in accuracy and loss across the federated rounds. The global model's performance, as evaluated on aggregated results from the clients, was plotted across all three rounds.

5.1 Global Accuracy per Round

This plot shows the improvement of the global model's accuracy over the course of training. The upward or steady trend reflects the effectiveness of the federated aggregation in capturing patterns from distributed data.

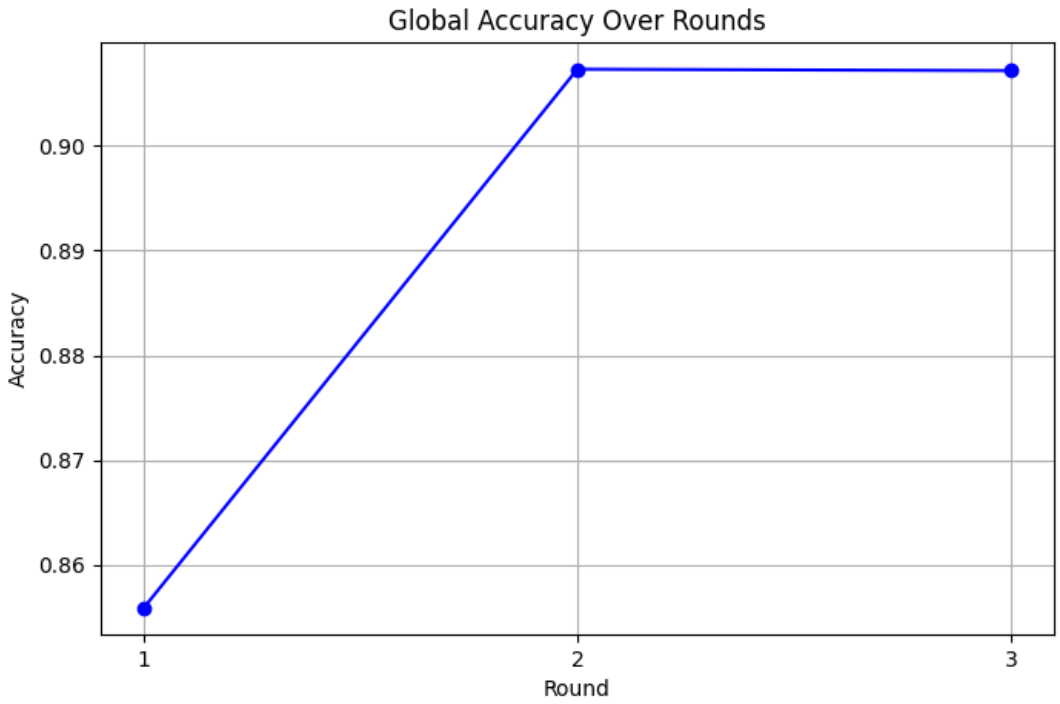


Figure 5.1: Global Accuracy

5.2 Global Loss per Round

A consistent or gradually decreasing loss indicates that the model is converging and learning effectively across rounds. Sudden spikes might suggest outlier updates or local overfitting, but in this simulation, loss generally remained stable or reduced.

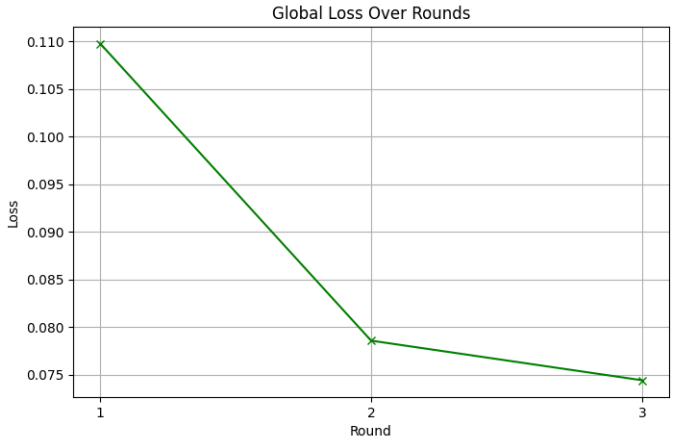


Figure 5.2: Global Loss

5.3 Client-wise Training and Validation Accuracy

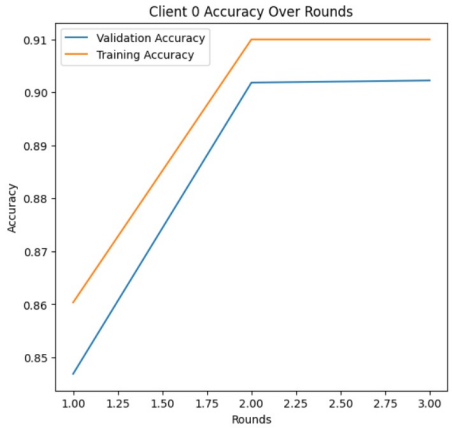
Each client trained its local model on its own dataset during every round. The training and validation accuracies for each client were tracked and plotted separately:

Training Accuracy

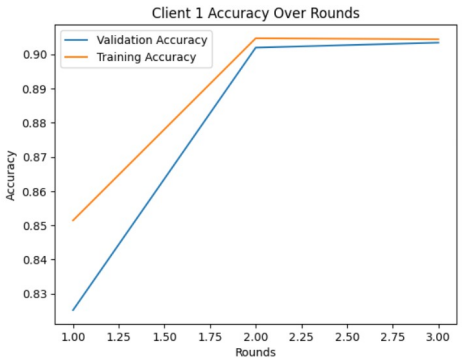
Training accuracy on both clients was generally higher than validation accuracy, which is expected as models fit better to training data. As the model is quite simple and there are not many features, accuracy remained relatively stable over multiple rounds. This behavior mirrors real-world scenarios where models may slightly overfit local data.

Validation Accuracy

Validation accuracy served as a better indicator of the model's generalization performance on unseen data. The trends were stable across the three rounds, suggesting that even with a minimal number of clients and simple features, the model was able to generalize moderately well.



(a) Client 0



(b) Client 1

Figure 5.3: Training and validation accuracy for Client 0 and Client 1

These results demonstrate how the model incrementally improved its understanding of the task through federated learning. Despite being trained on only a fraction of the dataset and never accessing raw data centrally, the system was able to produce a functioning model.

Chapter 6

Challenges Faced

During the course of this project, several practical challenges emerged, particularly related to hardware limitations. These challenges influenced the decisions and the scope of the implementation.

The main challenge encountered during the project was the limited hardware capability of the Raspberry Pi 3 B. Due to its 32-bit ARM architecture and only 1 GB of RAM, installing standard machine learning libraries such as TensorFlow or PyTorch was not feasible. This constraint necessitated a more hands-on approach, where the neural network was manually implemented using `NumPy` to ensure compatibility and efficiency. Even then, the limited memory posed occasional stability issues during training.

Another major challenge involved efforts to simulate real-time health data acquisition using sensors. The project initially aimed to interface an Arduino board with a heartbeat sensor to simulate real-time data acquisition. However, the sensor was unreliable—either failing to log any data or producing faulty readings. Additionally, a suitable blood pressure sensor was not available during the development period, which prevented the inclusion of one of the intended input features.

Due to these setbacks, the idea of using real-time physiological data had to be abandoned, and the project proceeded using standard datasets to simulate sensor outputs. While this shift limited the hardware integration aspect, it allowed the core concept of federated learning in an IoT-inspired setup to be fully explored.

Chapter 7

Conclusion and Future Work

In this research project, we explored the feasibility of implementing Federated Learning (FL) in a resource-constrained environment using low-powered edge devices like the Raspberry Pi. The system was designed as a proof of concept to demonstrate how FL could be applied to sensitive domains such as healthcare, where data privacy and decentralization are crucial.

Although only a limited number of features were used and the overall model complexity was intentionally kept low, the system successfully simulated a real-world FL environment. While the overall accuracy remained modest, the model's ability to flag potential risks—particularly false positives—should not be overlooked. In medical contexts, such early warnings can lead to precautionary actions that may help prevent serious health outcomes.

Moreover, the FL setup inherently enhances data privacy and broadens data representation, as the global model benefits from the diversity of locally trained models without accessing the raw data itself. This approach tackles a major challenge in healthcare AI: the shortage of large, diverse datasets for model training.

Future Work

Looking ahead, there are several promising directions to enhance and extend this work. One important improvement would be the integration of real-time physiological data through reliable sensors, which would bring the simulation closer to a practical IoT-healthcare deployment. Other possible directions include:

- **Using more powerful hardware:** Upgrading to newer Raspberry Pi models with better RAM and CPU could enable the use of machine learning frameworks like TensorFlow or PyTorch, allowing for more complex models and expanded feature sets.
- **Scaling the number of clients:** Increasing the number of participating edge devices would allow a deeper evaluation of the Flower framework's ability to manage communication, synchronization, and aggregation across a broader network.

- **Exploring non-IID data scenarios:** Simulating clients with non-identically distributed data would help in studying the robustness and adaptability of FL models when faced with real-world data imbalances and heterogeneity.

These extensions would provide valuable insights into both the practical deployment and theoretical capabilities of federated learning in decentralized, privacy-sensitive domains like healthcare.

References

[1] McMahan, H. B., Moore, E., Ramage, D., and y Arcas, B. A., 2016. "Federated learning of deep networks using model averaging". *CoRR,* **abs/1602.05629**.

[2] Sana, F., Isselbacher, E. M., Singh, J. P., Heist, E. K., Pathik, B., and Armoundas, A. A., 2020. "Wearable devices for ambulatory cardiac monitoring: Jacc state-of-the-art review". *Journal of the American College of Cardiology,* **75**(13), pp. 1582–1592.

[3] Rudraraju, S. R., Suryadevara, N. K., and Negi, A., 2023. "Heterogeneous sensor data acquisition and federated learning for resource constrained IOT devices—a validation". *IEEE Sensors Journal,* **23**(15), Jun., pp. 17602–17610.

[4] S, K. B., M, D., and K, N., 2022. "A federated learning based approach for heart disease prediction". In 2022 6th International Conference on Computing Methodologies and Communication (ICCMC), pp. 1117–1121.

[5] Bebortta, S., Tripathy, S. S., Basheer, S., and Chowdhary, C. L., 2023. "Fedehr: A federated learning approach towards the prediction of heart diseases in iot-based electronic health records". *Diagnostics,* **13**(20), p. 3166.

[6] J, A. J., Paulraj, G. J. L., M, G. R., Jebadurai, I. J., Janani, S. P., and Aarthi, M. S., 2024. "Edge-based heart disease prediction using federated learning". In 2024 International Conference on Cognitive Robotics and Intelligent Systems (ICC - ROBINS), pp. 294–299.

[7] Gupta, S., Kumar, P., Srivastava, N. V., Kumar, A., and Chaurasia, B. K., 2024. "Heart disease prediction using federated learning". In *Proceedings of International Conference on Recent Innovations in Computing*, Y. Singh, P. J. S. Gonçalves, P. K. Singh, and M. H. Kolekar, eds., Vol. 1196 of *Lecture Notes in Electrical Engineering*. Springer.

[8] Sonawane, J. S., and Patil, D. R., 2014. "Prediction of heart disease using multilayer perceptron neural network". In International Conference on Information Communication and Embedded Systems (ICICES2014), pp. 1–6.

Federated Learning for Privacy-Preserving Heart Disease Prediction

A report submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Alen Scaria

(Roll Number: 121CS0237)

based on research carried out

under the supervision of

Prof. Judhistir Mahapatro

May, 2025

Department of Computer Science and Engineering
National Institute of Technology Rourkela

Federated Learning for Privacy-Preserving Heart Disease Prediction

ORIGINALITY REPORT

12%	9%	11%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	web.realinfo.tv Internet Source	1%
2	www.mdpi.com Internet Source	1%
3	Edlira Martiri, Narasimha Rao Vajjhala, Fisnik Dalipi. "AI-Enabled Threat Intelligence and Cyber Risk Assessment", CRC Press, 2025 Publication	1%
4	www.coursehero.com Internet Source	1%
5	Vivek S. Sharma, Shubham Mahajan, Anand Nayyar, Amit Kant Pandit. "Deep Learning in Engineering, Energy and Finance - Principles and Applications", CRC Press, 2024 Publication	1%
6	fastercapital.com Internet Source	1%
7	Chhaya Gupta, Nasib Singh Gill, Preeti Gulia, Noha Alduaiji, J. Shreyas, Piyush Kumar Shukla. "Applying YOLOv6 as an ensemble federated learning framework to classify breast cancer pathology images", Scientific Reports, 2025 Publication	1%

8	Vieira, Pedro Manuel Ribeiro. "A Federated Learning Approach for Data Privacy in Healthcare Applications.", Instituto Politecnico do Porto (Portugal), 2024 Publication	<1 %
9	"Advanced Intelligent Computing Technology and Applications", Springer Science and Business Media LLC, 2024 Publication	<1 %
10	Hichem Metmer, Xiaoshan Yang. "FedMRG: federated medical report generation via text-aware learning rate adjustment and multi-level prototype collaboration", Multimedia Systems, 2025 Publication	<1 %
11	Shubhi Shukla, Suraksha Rajkumar, Aditi Sinha, Mohamed Esha, Konguvel Elango, Vidhya Sampath. "Federated learning with differential privacy for breast cancer diagnosis enabling secure data sharing and model integrity", Scientific Reports, 2025 Publication	<1 %
12	www.nature.com Internet Source	<1 %
13	Hao Meng, Qiang Zhan, Changwei Ji, Jinxin Yang, Shuofeng Wang. "Identification, prediction and classification of hydrogen-fueled Wankel rotary engine knock by data-driven based on combustion parameters", Energy, 2024 Publication	<1 %
14	docsplayer.org Internet Source	<1 %

15	medium.com Internet Source	<1 %
16	Liu, Yuanhao. "Innovative Deep Learning Models and Applications for Diverse Statistical Problems.", Rutgers The State University of New Jersey, School of Graduate Studies, 2024 Publication	<1 %
17	Biswadip Basu Mallik, Gunjan Mukherjee, Rahul Kar, Aryan Chaudhary. "Deep Learning Concepts in Operations Research", Routledge, 2024 Publication	<1 %
18	ijrpr.com Internet Source	<1 %
19	Pradeep Singh, Balasubramanian Raman. "Deep Learning Through the Prism of Tensors", Springer Science and Business Media LLC, 2024 Publication	<1 %
20	mssc.mu.edu Internet Source	<1 %
21	www.econstor.eu Internet Source	<1 %
22	www.kaell.se Internet Source	<1 %
23	aiforsocialgood.ca Internet Source	<1 %
24	papyrus.bib.umontreal.ca Internet Source	<1 %
25	vec.wikipedia.org Internet Source	<1 %

26	www.maximizemarketresearch.com Internet Source	<1%
27	Mei Cao, Yujie Zhang, Zezhong Ma, Mengying Zhao. " : Class-aware Client Selection for Effective Aggregation in Federated Learning ", High-Confidence Computing, 2022 Publication	<1%
28	vit.edu.in Internet Source	<1%

Exclude quotes	On	Exclude matches	< 15 words
Exclude bibliography	On		