HEART DISEASE PREDICTION USING FEDERATED LEARNING ALGORITHMS

MINI PROJECT REPORT

Submitted by

KAMAL M

KRISHNA K

SARAVANAKUMAR T

VISHVA K

in partial fulfillment of the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND BUSINESS SYSTEMS

K. RAMAKRISHNAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

SAMAYAPURAM,TRICHY

ANNAUNIVERSITY

CHENNAI 600 025

DECEMBER 2024

HEART DISEASE PREDICTION USING FEDERATED LEARNING ALGORITHMS

UCB1512 MINI PROJECT REPORT

Submitted by

KAMAL M	**(8115U22CB024)**
KRISHNA K	**(8115U22CB032)**
SARAVANAKUMAR T	**(8115U22CB048)**
VISHVA K	**(8115U22CB061)**

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND BUSINESS SYSTEMS

Under the Guidance of

Mrs.B.SATHIYA, B.E, M.Tech.,

Department of Computer Science and Business Systems

K. RAMAKRISHNAN COLLEGE OF ENGINEERING

COMPUTER SCIENCE AND BUSINESS SYSTEMS

K.RAMAKRISHNAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

Under

ANNA UNIVERSITY, CHENNAI

K.RAMAKRISHNAN COLLEGE OF ENGINEERING (AUTONOMOUS)
Under
ANNA UNIVERSITY, CHENNAI

BONAFIDE CERTIFICATE

Certified that this project report **"HEART DISEASE PREDICTION USING FEDERATED LEARNING ALGORITHMS"** is the Bonafide work of **KAMAL M (8115U22CB024), KRISHNA K (8115U22CB032), SARAVANAKUMAR T (8115U22CB048), VISHVA K (8115U22CB061)** who carried out the project work under my supervision.

Dr.J.SASIDEVI, M.E., Ph.D.,	**Mrs.B.SATHIYA, B.E., M.Tech.,**
Associate Professor,	**Assistant Professor,**
HEAD OF THE DEPARTMENT,	**SUPERVISOR,**
Department of Computer Science and Business Systems	Department of Computer Science and Business Systems
K.Ramakrishnan College of Engineering, (Autonomous)	K.Ramakrishnan College of Engineering, (Autonomous)
Samayapuram, Trichy-621112.	Samayapuram, Trichy-621112.

SIGNATURE OF INTERNAL EXAMINER	**SIGNATURE OF EXTERNAL EXAMINER**
NAME:	**NAME:**
DATE:	**DATE:**

K.RAMAKRISHNAN COLLEGE OF ENGINEERING (AUTONOMOUS)
Under
ANNA UNIVERSITY, CHENNAI

ACKNOWLEDGEMENT

We thank the almighty GOD, without whom it would not have been possible for us to complete our project.

We wish to address our profound gratitude to **Dr.K.RAMAKRISHNAN,** Chairman, K.Ramakrishnan College of Engineering (Autonomous), who encouraged and gave us all help throughout the course.

We express our hearty gratitude and thanks to our honourable and grateful Executive Director **Dr.S.KUPPUSAMY, B.Sc., MBA., Ph.D.,** K.Ramakrishnan College of Engineering (Autonomous).

We are glad to thank our Principal **Dr.D.SRINIVASAN, M.E., Ph.D., FIE., MIIW., MISTE., MIS, AE., C.Eng.,** K.Ramakrishnan College of Engineering (Autonomous), for giving us permission to carry out this project.

We wish to convey our sincere thanks to **Dr.J.SASIDEVI, M.E., Ph.D.,**Head of the Department in Computer Science and Business Systems, K.Ramakrishnan College of Engineering (Autonomous), for giving us constant encouragement and advice throughout the course.

We are grateful to **Mrs.B.SATHIYA, B.E., M.Tech.,** Assistant Professor of Department of Computer Science and Business Systems, K.Ramakrishnan College of Engineering (Autonomous), for her guidance and valuable suggestions during the course of study.

We sincerely thank, **Dr.P.SHANMUGA PRIYA, M.E., Ph.D.,** for outstanding leadership and dedication throughout coordinating the project

Finally, we sincerely acknowledge in no less terms all our staff members, colleagues, our parents and friends for their cooperation and help at various stages of this project work.

K.RAMAKRISHNAN COLLEGE OF ENGINEERING
(AUTONOMOUS)
Under
ANNA UNIVERSITY, CHENNAI

DECLARATION BY THE CANDIDATE

I declare that to the best of my knowledge the work reported here in has been composed solely by myself and that it has not been in whole or in part in any previous application for a degree.

Submitted for the Mini project Viva-Voce held at K.Ramakrishnan College of Engineering on _________

SIGNATURE OF THE CANDIDATE

(KAMAL M)

K.RAMAKRISHNAN COLLEGE OF ENGINEERING
(AUTONOMOUS)
Under
ANNA UNIVERSITY, CHENNAI

DECLARATION BY THE CANDIDATE

I declare that to the best of my knowledge the work reported here in has been composed solely by myself and that it has not been in whole or in part in any previous application for a degree.

Submitted for the Mini project Viva-Voce held at K.Ramakrishnan College of Engineering on ________

SIGNATURE OF THE CANDIDATE

(KRISHNA K)

K.RAMAKRISHNAN COLLEGE OF ENGINEERING
(AUTONOMOUS)
Under
ANNA UNIVERSITY, CHENNAI

DECLARATION BY THE CANDIDATE

I declare that to the best of my knowledge the work reported here in has been composed solely by myself and that it has not been in whole or in part in any previous application for a degree.

Submitted for the Mini project Viva-Voce held at K.Ramakrishnan College of Engineering on ________

SIGNATURE OF THE CANDIDATE

(SARAVANAKUMAR T)

K.RAMAKRISHNAN COLLEGE OF ENGINEERING
(AUTONOMOUS)
Under
ANNA UNIVERSITY, CHENNAI

DECLARATION BY THE CANDIDATE

I declare that to the best of my knowledge the work reported here in has been composed solely by myself and that it has not been in whole or in part in any previous application for a degree.

Submitted for the Mini project Viva-Voce held at K.Ramakrishnan College of Engineering on _

SIGNATURE OF THE CANDIDATE

(VISHVA K)

ABSTRACT

Heart disease remains a significant global health challenge, responsible for millions of deaths annually. Early detection is critical for effective intervention, but the healthcare sector faces unique challenges in developing predictive models due to privacy concerns, data fragmentation, and stringent regulations like GDPR and HIPAA. This project addresses these challenges through the development of a federated learning-based heart disease prediction system, which ensures robust performance while preserving patient data privacy. The project utilizes a neural network model, developed with TensorFlow and Keras, to predict the likelihood of heart disease based on patient data. Unlike traditional centralized approaches, the federated learning paradigm enables multiple clients (such as hospitals or medical institutions) to collaboratively train a shared global model without transferring raw data. Each client processes data locally and shares only encrypted model updates with a central server. The system employs cryptographic techniques to secure communication, ensuring that sensitive medical information remains confidential. A federated averaging algorithm aggregates these updates efficiently, creating a global model that benefits from the collective knowledge of all clients while preserving individual privacy.Experimental results demonstrate the system's ability to handle diverse datasets and achieve exceptional performance metrics. Specifically, the model attained an accuracy of 98.36%, highlighting its effectiveness in predicting heart disease. This result underscores the potential of federated learning to overcome the limitations of centralized data processing, such as risks of breaches, loss of ownership, and non-compliance with privacy standards. The scalable and adaptable architecture of this system ensures its applicability across heterogeneous data distributions, paving the way for secure, privacy preserving predictive healthcare solutions.

TABLE OF CONTENT

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 HEART DISEASE

The project focuses on using federated learning to predict heart disease by training a neural network model while ensuring data privacy and security. Federated learning allows multiple clients (such as hospitals or health devices) to collaboratively train a shared model without directly sharing sensitive data. In this project, the clients are integrated using cryptographic keys to ensure secure communication and data privacy throughout the process. The model is trained using TensorFlow and Keras, where the central server coordinates the training via the Federated Averaging algorithm, which aggregates updates from the clients and refines the global model.

The goal of this approach is to leverage data from multiple sources without compromising privacy, allowing for more accurate and generalized predictions of heart disease, while still complying with privacy regulations like HIPAA. This method enhances the security of sensitive medical data by ensuring that the data does not leave the client devices, which is particularly important in health-related applications.

1.2 SYMPTOMS

The symptoms of heart disease can vary depending on the type and severity of the condition. In the context of heart disease prediction, certain symptoms are often used as indicators to assess the likelihood of heart disease. Some of the common symptoms associated with heart disease include

1.2.1 Chest Pain (Angina)

A sensation of tightness, pressure, or pain in the chest, often triggered by physical activity or emotional stress.

1.2.2 Shortness of Breath

Difficulty breathing, especially during physical activity or even at rest, indicating possible heart failure or coronary artery disease.

1.2.3 Fatigue

Persistent tiredness or lack of energy, even after rest, which can be a sign of heart failure or arrhythmias.

1.2.4 Palpitations

Irregular, fast, or pounding heartbeats that may feel like fluttering or thumping in the chest, often related to arrhythmias.

1.2.5 Swelling (Edema)

Fluid buildup causing swelling in the legs, ankles, feet, or abdomen, often linked to heart failure.

1.2.6 Dizziness or Lightheadedness

A feeling of faintness or dizziness, which could indicate low blood pressure, arrhythmias, or poor circulation from heart disease.

Recognizing these symptoms early is crucial for diagnosing heart disease and taking preventive or corrective actions. Many of these symptoms, such as chest pain or shortness of breath, are often used in predictive models for heart disease to assess a patient's risk.

1.3 FEDERATED LEARNING

Federated Learning is an innovative machine learning approach that enables decentralized model training across multiple devices or clients without the need to share sensitive or private data. Unlike traditional machine learning, where data is centralized in a single server for training, federated learning allows models to be trained locally on each device, with only model updates (such as weights or gradients) being shared with a central server. This ensures that raw data remains on the local devices, significantly enhancing privacy and data security.

Federated learning is particularly valuable in scenarios where data privacy is a concern, such as in healthcare, finance, or mobile applications. For instance, hospitals can collaboratively train a predictive model for disease diagnosis without transferring patient data. Similarly, mobile devices can improve app performance (like predictive text or voice recognition) based on user data, without sending personal information to centralized servers.

The process involves multiple iterations where a central server sends a global model to local devices, which train the model on their local data. The model updates are then aggregated by the server to refine the global model. This iterative process continues, allowing the model to improve over time.

1.4 TECHNOLOGIES IN FEDERATED LEARNING

1.4.1 Machine Learning Frameworks

Frameworks like TensorFlow Federated (TFF) and PySyft are designed for decentralized and privacy-preserving training. Keras is used for neural network design, while Flower (FL) simplifies the orchestration of federated systems.

1.4.2 Cryptographic Techniques

Techniques like Secure Aggregation encrypt client updates to ensure privacy, while Homomorphic Encryption allows computations on encrypted data. Differential Privacy adds noise to updates to prevent data leakage, and PKI manages secure communication.

1.4.3 Communication Protocols

Efficient communication is achieved with gRPC for high-speed data exchange, WebSockets for real-time updates, and MQTT for lightweight messaging, especially in IoT setups.

1.4.4 Optimization and Algorithms

The Federated Averaging (FedAvg) algorithm combines client updates to create a global model, while FedOpt uses advanced optimizers like Adam for better results. Client Sampling ensures only a subset of clients train in each round, saving resources.

1.4.5 Data Handling

Federated learning keeps data local on devices, using federated data management techniques. It addresses challenges like non-IID data, where client data distributions differ significantly, mimicking real-world scenarios.

1.4.6 Cloud and Edge Computing

Training happens on edge devices like smartphones or IoT devices, while cloud platforms like Google Cloud or AWS aggregate model updates for global optimization.

1.5 ADVANTAGES OF FEDERATE LEARNING

Data Privacy and Security

Federated learning keeps data on client devices, ensuring sensitive information never leaves the source. This minimizes risks of data breaches and complies with privacy regulations like GDPR and HIPAA.

Decentralized Training

Models are trained across multiple devices without centralizing data, reducing the need for large-scale data transfers and central storage.

Scalability

Federated learning can scale across thousands or even millions of devices, making it ideal for applications involving diverse and widespread data sources like IoT devices or smartphones.

Personalization

Local training allows models to adapt to user-specific data, improving performance for individual clients while contributing to a global model.

Reduced Communication Costs

Instead of transferring raw data, only model updates are communicated to the central server, reducing bandwidth usage.

1.6 ANALYTICAL FRAMEWORK

Creating an analytical framework for federated learning involves understanding the key components and evaluating its performance, privacy, and efficiency. Here's a structured analytical framework:

System Design

Federated learning systems can use centralized (server-client), decentralized (peer-to-peer), or hybrid architectures. The design must consider communication protocols and hardware resources like edge devices and servers for computation and storage.

Data Distribution and Handling

Data is distributed across clients and may be IID (identically distributed) or non-IID (varied distribution). Privacy-preserving techniques like differential privacy and secure aggregation are used to handle data securely without centralization.

Learning Algorithms

Federated learning relies on algorithms like Federated Averaging (FedAvg) for aggregating local updates into a global model. Strategies like adaptive optimization and client sampling improve efficiency and personalization.

Communication Efficiency

Efficient communication minimizes bandwidth usage by transferring model updates instead of raw data. The frequency of updates and latency between clients and servers are key factors for optimization.

Security and Privacy

Cryptographic techniques, such as homomorphic encryption and secure aggregation, ensure data privacy. The system must also defend against adversarial attacks and prevent data leakage during training.

CHAPTER 2

LITERATURE SURVEY

TITLE: A Federated Learning-Based Approach for Heart Disease Prediction

AUTHOR : A. S. R. Anjaneyulu et al.

YEAR OF PUBLICATION : 2021

Summary: This research explores the potential of federated learning (FL) for heart disease prediction in healthcare settings, where data privacy is a major concern. The study introduces the concept of federated averaging, which allows multiple institutions or hospitals to train local models on their data without sharing sensitive patient information. The local model updates are aggregated to create a global model that can effectively predict heart disease risks while ensuring the confidentiality of medical data. The authors highlight the advantages of using FL over traditional centralized data-sharing models, as it ensures that sensitive information remains protected.

Contribution: This paper makes a significant contribution by demonstrating the practicality of FL in real-world healthcare applications, where privacy laws like HIPAA (Health Insurance Portability and Accountability Act) prevent the centralized sharing of patient data. The study shows that FL can be used to develop predictive models that adhere to privacy standards while still providing reliable outcomes for heart disease diagnosis

TITLE: Federated Learning in Predicting Heart Disease

AUTHOR : H. Z. Huang et al.

YEAR OF PUBLICATION : 2023

Summary: This study focuses on applying federated learning (FL) to predict heart disease by utilizing data from multiple decentralized hospitals. The authors examine how federated learning can address privacy concerns while still achieving accurate predictions for heart disease diagnosis. By training models locally at each institution and sharing only model parameters (rather than raw data), the study ensures that patient data stays private. The authors perform a comparison between federated and centralized learning, noting that while federated learning may have slightly lower performance in some cases, it is still a valuable technique for situations where data privacy is crucial.

Contribution: The study emphasizes the trade-off between privacy and performance and explores techniques to minimize the performance gap between centralized and federated learning models. By applying FL to heart disease prediction, this paper opens avenues for using machine learning in healthcare without compromising data security

TITLE: FedEHR: A Federated Learning Approach towards the Prediction of Heart Diseases in IoT-Based Electronic Health Records

AUTHOR : Subhranshu Sekhar Tripathy et al.

YEAR OF PUBLICATION : 2023

Summary: The paper introduces a novel approach by combining federated learning with IoT (Internet of Things) devices and Electronic Health Records (EHRs) to predict heart disease. The study demonstrates how wearable devices (such as smartwatches) can collect real-time health data, which is then incorporated into EHR systems. This data is used to train federated learning models, ensuring that no patient data is shared across institutions. The federated learning framework aggregates local models from various IoT devices and healthcare centers, improving prediction accuracy while preserving privacy.

Contribution: The integration of IoT with federated learning is particularly noteworthy in this paper, as it addresses the challenge of collecting real-time, continuous health data for heart disease prediction. The combination of high-dimensional sensor data and EHRs in a federated learning setting allows for personalized and accurate heart disease predictions. This paper illustrates how healthcare systems can leverage both IoT and FL for a privacy-preserving, scalable prediction model

TITLE : Enhancing Heart Disease Prediction with Federated Learning and Blockchain Integration

AUTHOR : Chaosheng Hu et al.

YEAR OF PUBLICATION : 2024

Summary: This paper takes an innovative approach by combining federated learning with blockchain technology for heart disease prediction. The authors utilize the TabNet model, a hybrid of tree-based models and deep learning techniques, within a federated learning framework to predict heart disease across decentralized institutions. Blockchain is integrated to ensure that all interactions between participants are transparent and secure, enabling immutable records of model updates and ensuring accountability.

Contribution: The integration of blockchain provides a unique solution to the challenges of data integrity and transparency in federated learning. By combining blockchain with FL, this paper creates a robust and secure framework for heart disease prediction. The use of smart contracts for automating governance and ensuring data privacy is another key innovation in this work. The study also demonstrates that this hybrid system can maintain high predictive accuracy while improving model transparency and preventing fraudulent or erroneous model updates

TITLE: Federated Learning for Heart Disease Prediction: Challenges and Opportunities

AUTHOR : M. S. Malvankar et al.

YEAR OF PUBLICATION : 2024

Summary: This review paper surveys the challenges and opportunities of using federated learning in healthcare, specifically in predicting heart disease. The authors discuss how FL can enable collaboration among multiple healthcare institutions without the need to centralize sensitive data, addressing privacy concerns like data leaks or breaches. The paper also explores the integration of federated learning with other privacy-preserving techniques, such as differential privacy, to enhance the robustness of heart disease prediction models.

Contribution: The review provides a comprehensive understanding of the current state of FL in healthcare, outlining both its potential and the barriers to widespread adoption. By exploring different techniques that can be used alongside FL (e.g., differential privacy), this paper offers valuable insights for improving the scalability and effectiveness of heart disease prediction systems

CHAPTER 3

SYSTEM ANALYSIS

In a system analysis for heart disease prediction using federated learning (FL), the goal is to create a predictive model that can accurately identify individuals at risk of heart disease without the need to centralize sensitive medical data. Here's a breakdown of how your system can be analyzed, including components such as federated learning algorithms, cryptographic security, and the use of neural networks with TensorFlow and Keras.

System Components

1. Federated Learning Framework

- **Federated Learning Algorithm**: In our system, the core algorithm for training the model is federated averaging. This approach allows the model to be trained across multiple decentralized clients (hospitals or medical institutions) without the need to share raw data. Each client (a healthcare facility or hospital) trains the model locally using its own data, and only the model updates (gradients) are sent to a central server, which aggregates them to improve the global model. This ensures data privacy and efficiency in training, especially when sensitive patient information is involved.
- **Federated Averaging (FedAvg)**: The FedAvg algorithm is typically used in federated learning for model aggregation. Clients train their local models and send the parameter updates (gradients) to the central server. The server averages these gradients and updates the global model, which is then sent back to the clients for further training. This approach helps in reducing the communication overhead and ensures the privacy of the local data by not sharing it directly.

2. Neural Network Model

- **Architecture**: A neural network is used for heart disease prediction. The architecture may include fully connected layers, convolutional layers (if you're using medical imaging), or recurrent layers (if working with time-series data like patient medical records). TensorFlow and Keras are suitable frameworks for implementing deep learning models due to their flexibility and ease of use. You might use a multi-layer perceptron (MLP) or more advanced models like CNNs (Convolutional Neural Networks) for data involving imaging (e.g., ECG signals or angiograms)

- **Training Process**: Each client locally trains the neural network model using its own dataset, which may include various features like blood pressure, cholesterol levels, ECG data, or patient demographics. These models are trained until they converge, and model weights (not raw data) are sent back to the server for aggregation.

3. Cryptographic Integration

- **Cryptography Keys**: To ensure the privacy and security of the data and model updates, cryptographic keys can be used to secure the communication between clients and the central server. This could involve techniques like Homomorphic Encryption or Secure Multi-Party Computation (SMPC), which allow computations to be performed on encrypted data. This means that even the server doesn't have access to the raw data but can still participate in the aggregation process. This is essential in maintaining patient confidentiality and complying with data protection regulations (e.g., HIPAA in the U.S.).
- **Data Privacy**: One of the major strengths of federated learning is that it prevents raw medical data from being shared across clients, addressing concerns about data security and privacy. Instead of sharing the actual dataset, only model updates are transmitted, encrypted, and aggregated. This limits the risk of unauthorized access to sensitive patient information.

4. TensorFlow and Keras

- **Model Development**: TensorFlow and Keras are popular frameworks for deep learning that allow you to quickly design, train, and evaluate complex models. TensorFlow, with its ability to scale across distributed systems, works well for federated learning, as it supports operations on local devices (clients) and aggregation on central servers.
- **Implementation**: Using Keras, you can build a neural network with layers like Dense, Dropout, and BatchNormalization to ensure that the model is robust and can generalize well to new data. You may also implement techniques like early stopping or learning rate schedules to optimize the training process and prevent overfitting.

5. Heart Disease Prediction

Data Utilization: Data used in heart disease prediction can include structured data (e.g., patient medical records, lab results) and unstructured data (e.g., ECG or other diagnostic imaging data). The federated learning model can be adapted to handle both types of data by using multi-modal neural network architectures that combine different types of input (e.g., numeric data, images).

Accuracy and Validation: Once the model is trained across clients, it will be validated using performance metrics like accuracy, precision, recall, and F1-score to ensure that it can effectively predict heart disease. Performance can also be compared across clients to ensure that the global model generalizes well to different types of data from different regions.

3.1 EXISTING SYSTEM

While there isn't a specific, widely-recognized commercial system solely dedicated to heart disease prediction using federated learning, several research papers and prototypes have explored this concept. Here's a general overview of the system and a diagram illustrating its components

3.1.1 Understanding the Concept

Federated Learning (FL) is a machine learning technique that enables multiple clients (like hospitals, clinics) to collaborate on training a shared model without sharing their raw data. This approach preserves data privacy while improving model accuracy.

3.1.2 System Components

1. **Data Sources**
   - Multiple healthcare institutions (hospitals, clinics)
   - Each institution possesses its own patient data, including medical records, lab results, and demographic information.
2. **Local Model Training**
   - Each institution trains a neural network model (e.g., CNN, RNN) on its local data to predict heart disease risk.
   - The model architecture can vary based on the specific data and requirements.

3. **Model Aggregation**
   - Encrypted model updates from each institution are sent to a central server.
   - The server aggregates these updates using secure multi-party computation (SMPC) techniques to preserve data privacy.
4. **Global Model Update**
   - The aggregated model updates are used to update the global model.
   - This global model represents the collective knowledge from all participating institutions.
5. **Model Sharing**
   - The updated global model is shared with all institutions.
   - Each institution can then fine-tune the global model on its local data to improve its performance.

3.1.3 SYSTEM DIAGRAM

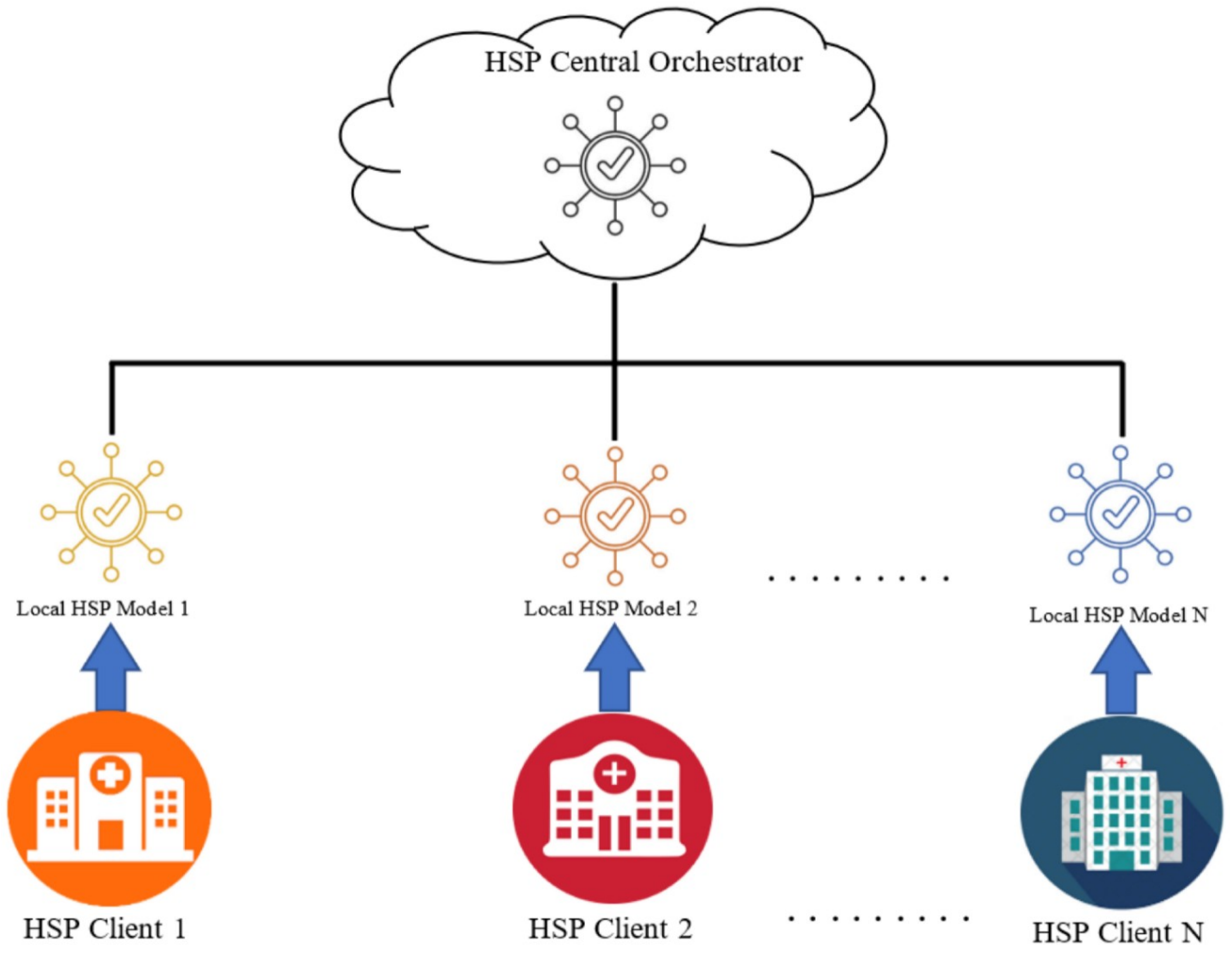


Fig 3.1 Existing system diagram

3.1.4 Key Considerations

- **Data Privacy:** Ensuring the privacy of patient data is paramount. Encryption and secure aggregation techniques are crucial.
- **Model Accuracy:** The accuracy of the global model depends on the quality of the local models and the effectiveness of the aggregation process.
- **Communication Efficiency:** Efficient communication between institutions and the server is essential to minimize latency and computational overhead.
- **Ethical Considerations:** Fair and unbiased model development is crucial to avoid discriminatory outcomes.

While specific implementations may vary, this general framework provides a solid foundation for federated learning in heart disease prediction. By leveraging the collective knowledge of multiple institutions while preserving data privacy, this approach has the potential to improve the accuracy and effectiveness of heart disease prediction models.

3.2 PROPOSED SYSTEM

This proposed system aims to leverage Federated Learning (FL) to train a robust and privacy-preserving heart disease prediction model. The system will involve multiple clients (hospitals, clinics) collaborating to train a shared model without sharing their raw patient data. This ensures data privacy while improving model accuracy.

3.2.1 Understanding Federated Learning

Federated Learning (FL) is a machine learning technique that enables multiple clients (like hospitals, clinics) to collaborate on training a shared model without sharing their raw data. This approach preserves data privacy while improving model accuracy.

3.2.2 Key Components of the System

1. Data Collection and Preprocessing

- **Data Sources**
  - Electronic Health Records (EHRs) from various healthcare institutions
  - Public health datasets

- **Data Preprocessing**
  - Cleaning: Handling missing values, outliers, and inconsistencies
  - Feature Engineering: Creating new features (e.g., calculating risk scores)
  - Normalization: Scaling features to a common range (e.g., 0-1)

2. Model Architecture

- **Neural Network:** A deep neural network (e.g., CNN, RNN) can be used to extract complex patterns from medical data.
- **Model Design:** The architecture should be suitable for heart disease prediction, handling numerical and categorical features.

3. Federated Learning Framework

- **Local Training:** Each client trains a local model on its own data.
- **Model Update:** Clients send encrypted model updates to the central server.
- **Model Aggregation:** The server aggregates the encrypted updates and updates the global model.
- **Model Sharing:** The updated global model is shared with the clients.

4. Model Training and Evaluation

- **Local Training:** Clients use optimization algorithms (e.g., SGD) to train their models.
- **Model Aggregation:** The server aggregates encrypted updates using secure multi-party computation (SMPC).
- **Global Model Update:** The server updates the global model.
- **Model Evaluation:** The global model is evaluated on a validation dataset.

3.2.4 SYSTEM DIAGRAM

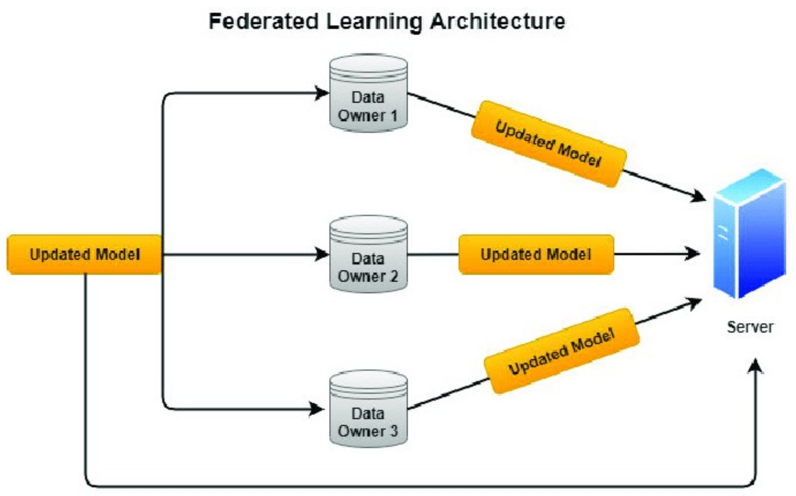


Fig 3.2 Proposed system diagram

3.2.5 Security and Privacy Considerations

- **Encryption:** Strong encryption (e.g., AES) protects data during transmission and storage.
- **Secure Aggregation:** SMPC techniques ensure privacy during model aggregation.
- **Differential Privacy:** Adding noise to model updates can further enhance privacy.

Ethical Considerations

- **Fairness:** Train the model on diverse data to avoid bias.
- **Transparency:** Make the model's decision-making process understandable.
- **Accountability:** Ensure accountability for the model's performance and impact.

CHAPTER 4

SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

The system architecture for heart disease prediction using federated learning is designed to integrate decentralized data sources, ensure data privacy, and enable collaborative model training. Below is a detailed system architecture divided into components and their interactions.

4.1.1 Components of the System

1. **Clients (Local Nodes)**
    - Represent data owners, such as hospitals or medical institutions.
    - Each client has its local dataset with patient records (e.g., age, cholesterol, blood pressure, etc.).
    - Performs local model training on patient data.
    - Encrypts model updates using cryptographic keys before sharing them.
2. **Central Server (Aggregator)**
    - Orchestrates the federated learning process.
    - Receives encrypted model updates from all clients.
    - Aggregates updates using the **Federated Averaging Algorithm** to update the global model.
    - Distributes the updated global model back to all clients.
3. **Cryptographic Infrastructure**
    - Implements secure encryption protocols (e.g., RSA, homomorphic encryption, or AES) for model update communication.
    - Ensures secure key generation, exchange, and management for privacy-preserving interactions between clients and the central server.
4. **Neural Network Model**
    - A classification model designed to predict heart disease based on patient features.
    - Built using TensorFlow and Keras, the model consists of:
        - Input layer for patient features.
        - Hidden layers with activation functions for pattern recognition.
        - Output layer for binary classification (heart disease: yes/no).

4.1.2 Key Functionalities of the Architecture

1. **Federated Learning Workflow**
    - **Initialization:** The central server initializes a global model and shares it with all clients.
    - **Local Training:** Each client trains the model on its private dataset and computes updates.
    - **Encryption:** Clients encrypt their updates using cryptographic techniques.
    - **Aggregation:** The server aggregates encrypted updates using the Federated Averaging Algorithm.
    - **Iteration:** The updated global model is shared with clients for further local training in subsequent rounds.
2. **Privacy-Preserving Mechanisms**
    - Local training ensures raw data never leaves the client devices.
    - Encrypted communication prevents interception of sensitive information.
3. **Global Model Deployment**
    - Once the global model converges, it is deployed to clients for heart disease prediction tasks.

4.1.3 Detailed Architecture Diagram

The system architecture can be divided into client-side, server-side, and the communication layer:

1. **Client-Side (Local Node)**
    - **Input Data:** Patient-specific features (e.g., age, cholesterol, blood pressure).
    - **Local Training Module**
        - Neural network model implementation.
        - Training logic using TensorFlow/Keras.
    - **Encryption Module:** Encrypts model updates using cryptographic keys.
    - **Communication Module:** Transmits encrypted updates to the central server.
2. **Server-Side (Central Node)**
    - **Initialization Module**
        - Generates and distributes the initial global model.
    - **Aggregation Module**
        - Implements the Federated Averaging Algorithm to combine client updates.
        - Updates the global model based on aggregated updates.

- **Distribution Module**
    - Sends the updated global model to clients after aggregation.

3. **Communication Layer**
    - Secure transmission of encrypted updates and global model weights between clients and the central server.
    - Utilizes secure protocols like HTTPS or TLS for transport security.

4.1.4 Federated Averaging Algorithm.

1. Local Update: Each client, denoted by i, updates its local model parameters w_i using a stochastic gradient descent (SGD) step:

w_i^(t+1) = w_i^(t) - η * ∇f_i(w_i^(t), x_i)

where:

- w_i^(t): Model parameters of client i at round t.
- η: Learning rate.
- ∇f_i(w_i^(t), x_i): Gradient of the local loss function f_i with respect to w_i^(t), calculated using a batch of data x_i from client i.

2. Global Aggregation: The server aggregates the updated model parameters from all clients to form a new global model:

w^(t+1) = (1/n) * Σ(w_i^(t+1))

where:

- w^(t+1): Global model parameters at round t+1.
- n: Number of clients.
- Σ(w_i^(t+1)): Sum of the updated model parameters from all clients.

Key Points:

- **Local Training:** Each client trains its model independently on its local data.
- **Parameter Sharing:** The server periodically broadcasts the global model parameters to all clients.
- **Model Aggregation:** The server averages the updated model parameters from all clients to form a new global model.
- **Iterative Process:** This process repeats for multiple rounds, allowing the global model to improve over time.

4.1.5 Tools and Frameworks

- **TensorFlow/Keras:** For building and training the neural network model.
- **TensorFlow Federated (TFF):** For implementing federated learning workflows.
- **Cryptographic Libraries:** PyCryptodome or similar for encryption of model updates.
- **Communication Protocols:** Secure communication using HTTPS or TLS.

4.1.6 SYSTEM ARCHITECTURE DIAGRAM

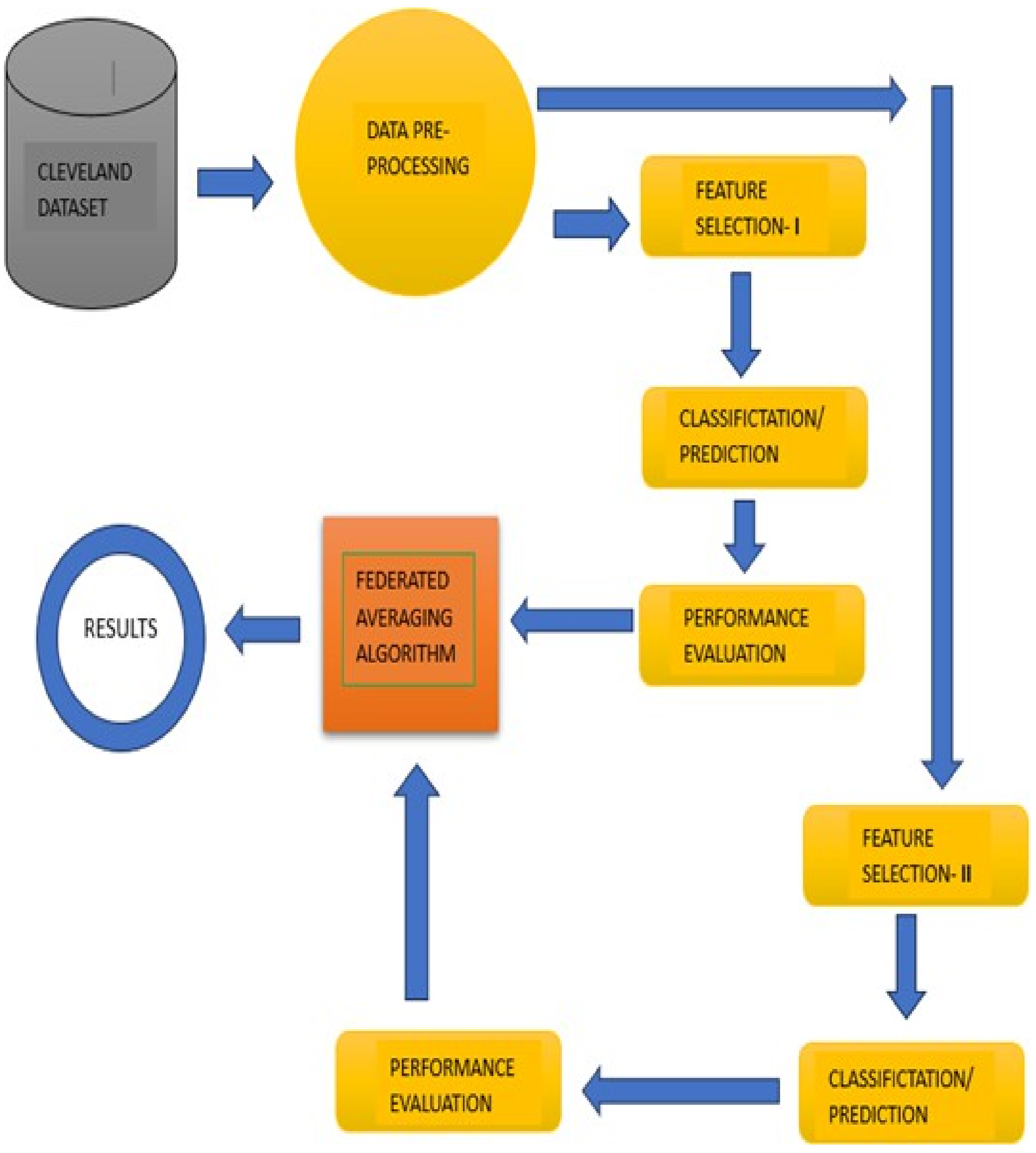


Fig 4.1 System Architecture diagram

4.2 UML DIAGRAM

ACTIVITY DIAGRAM

This activity diagram illustrates the workflow of a federated learning system for heart disease prediction:

1. **Data Collection and Preprocessing**
    - Clients collect and preprocess heart disease data from various sources (e.g., hospitals, clinics, wearable devices).
    - Data is cleaned, normalized, and formatted for model training.
2. **Local Model Training**
    - Each client trains a neural network model on its local data using TensorFlow and Keras.
    - Model architecture and hyperparameters are optimized to achieve the best performance.
3. **Model Update Encryption**
    - Clients encrypt their model updates using cryptographic techniques to ensure secure communication.
    - Encryption protects sensitive information from unauthorized access.
4. **Secure Communication**
    - Encrypted model updates are securely transmitted to the central server.
    - Secure communication protocols are employed to protect data integrity and confidentiality.
5. **Model Aggregation**
    - The central server decrypts and aggregates the model updates using federated averaging.
    - Federated averaging ensures privacy by combining model updates without revealing individual client data.
6. **Global Model Update**
    - The aggregated global model is encrypted and broadcast to all clients.
    - Encryption protects the global model during transmission.

7. **Local Model Update**

- Clients update their local models using the global model update.
- Local models are fine-tuned based on the global knowledge, improving their performance.

8. **Iterative Process**

- Steps 3-7 are repeated for multiple rounds to refine the global model.
- The iterative process continues until convergence or a predefined number of rounds.

This federated learning approach enables collaborative training of a heart disease prediction model while preserving data privacy and security.

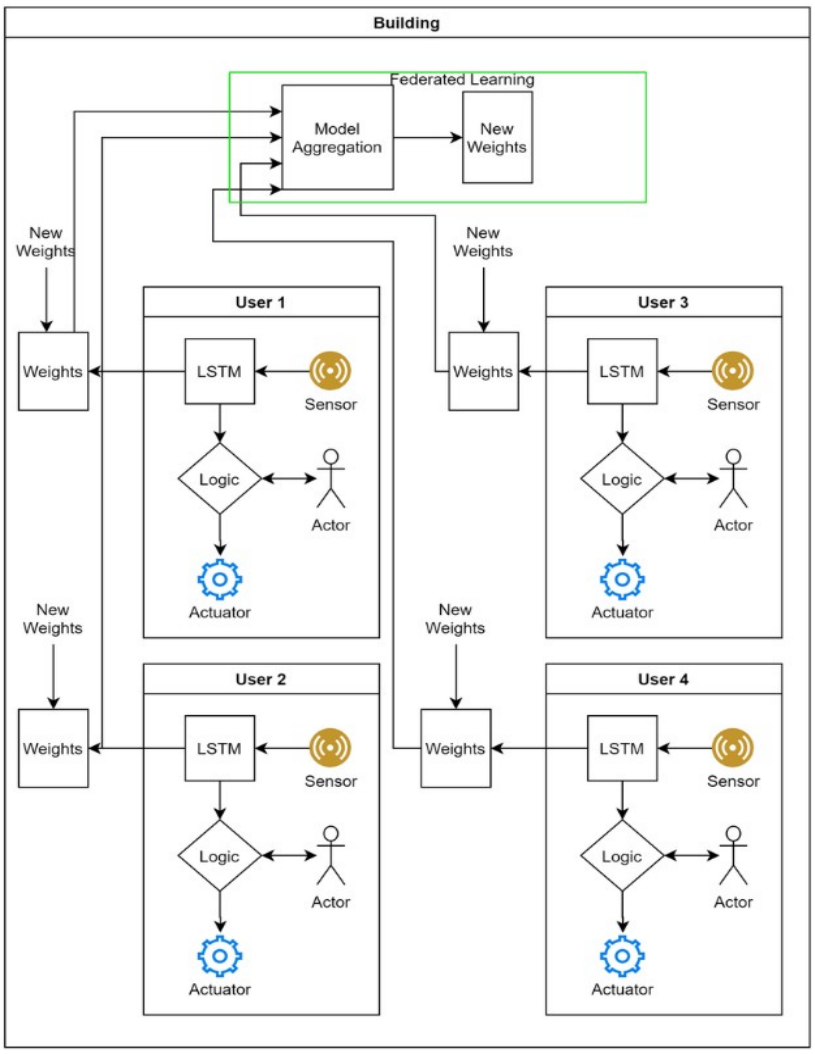


Fig 4.2 Activity Diagram

CHAPTER 5

MODULES AND DESCRIPTIONS

5.1 DATA PREPROCESSING

Data preprocessing and feature engineering are essential steps in developing a robust heart disease prediction model using federated learning. These steps ensure that raw patient data is transformed into a consistent and clean format suitable for training neural networks while maintaining data privacy. The preprocessing begins with handling missing values, where numerical fields like cholesterol and blood pressure are imputed using mean or median values, and categorical data is assigned appropriate encodings. Outliers in critical features such as age and cholesterol levels are detected and addressed using statistical methods like Z-score or interquartile range analysis to maintain data consistency.

Feature scaling is then applied to normalize numerical variables, ensuring that all features contribute equally to the model's training process. Standardization or MinMax scaling is commonly used to transform features like blood pressure and cholesterol levels into a uniform range. For categorical variables, encoding methods such as one-hot encoding or label encoding are employed to convert textual categories (e.g., gender or chest pain types) into numerical values that the neural network can process effectively.

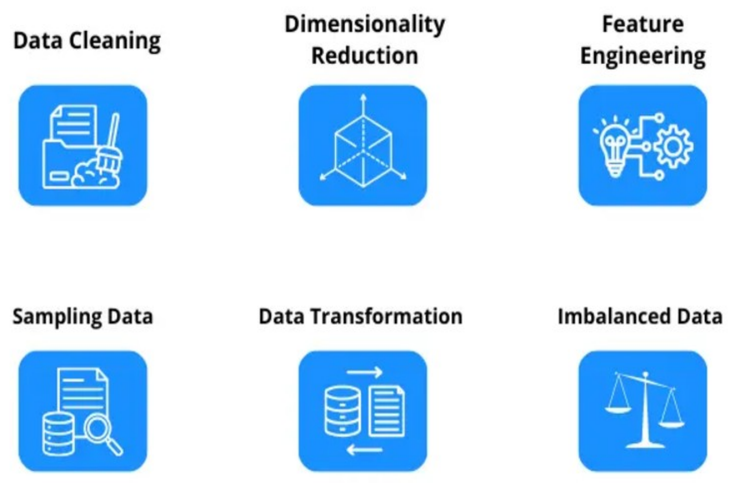


Fig 5.1 Data Preprocessing and Feature Engineering

5.2 MODEL SELECTION AND TRAINING

In the heart disease prediction system using federated learning, the process begins with selecting a robust and efficient neural network architecture. The model is designed to balance complexity and performance, typically comprising an input layer matching the number of features, two to three dense hidden layers with ReLU activation, and an output layer tailored for binary classification using a sigmoid activation function. Regularization techniques, such as dropout and L2 regularization, are incorporated to mitigate overfitting, particularly for smaller client datasets. The Adam optimizer is chosen for its adaptability and efficiency, paired with a binary cross-entropy loss function to optimize the model's performance.

Training is conducted locally at each client, where the global model, initialized by the server, is sent to clients for local updates. Each client trains the model on its private dataset, performing forward propagation, backpropagation, and weight updates. After training, the clients encrypt their model updates using cryptographic keys before transmitting them to the server. At the server, the updates are aggregated using the Federated Averaging Algorithm (FedAvg), which computes a weighted average of client updates based on their dataset sizes. The updated global model is then redistributed to clients, iterating this process until the model converges.

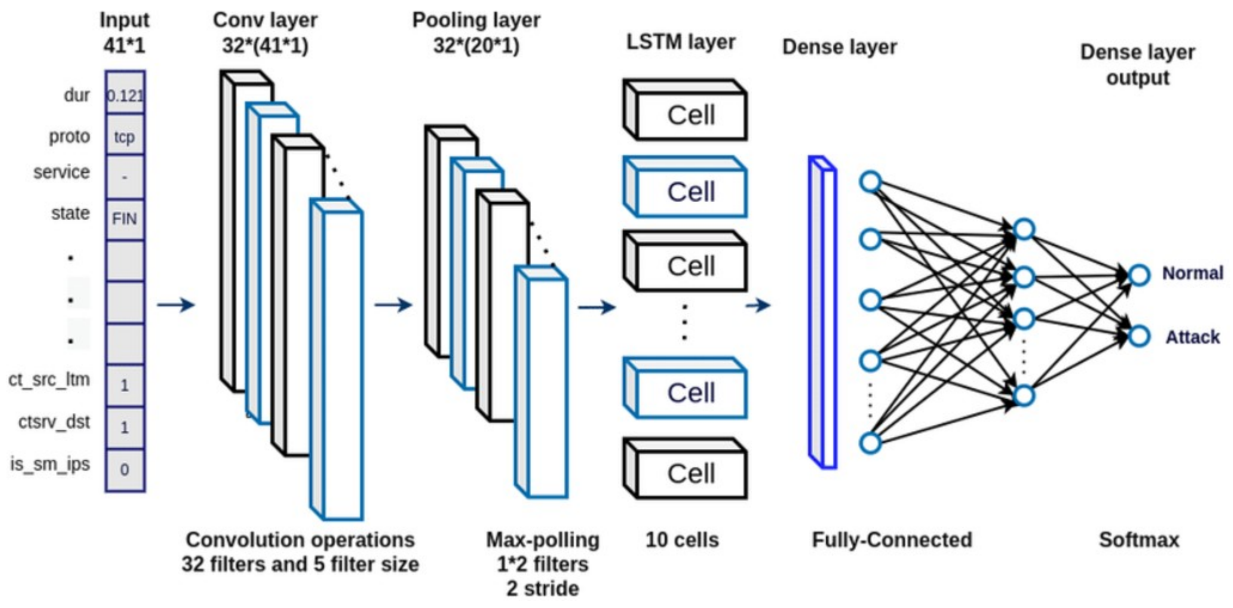


Fig 5.2 Model Selection And Training

5.3 MODEL EVALUATION AND INTERPRETATION

Model evaluation and interpretation are vital for ensuring the heart disease prediction model’s performance and reliability. Key evaluation metrics like accuracy, precision, recall, F1-score, and ROC-AUC provide a comprehensive view of the model’s strengths and weaknesses. Accuracy measures overall prediction correctness, while precision and recall focus on minimizing false positives and maximizing true positives, respectively. The F1-score combines precision and recall into a single metric, and ROC-AUC evaluates the model's ability to distinguish between classes, especially in imbalanced datasets.

Using Scikit-learn, these metrics can be easily computed, alongside a confusion matrix, which gives a deeper look into misclassified cases. Additionally, threshold tuning via the ROC curve allows for optimizing the trade-off between precision and recall, while techniques like SHAP values or permutation importance offer insights into the influence of each feature on predictions. Together, these evaluation methods ensure that the model is both accurate and interpretable, improving its reliability for real-world deployment.

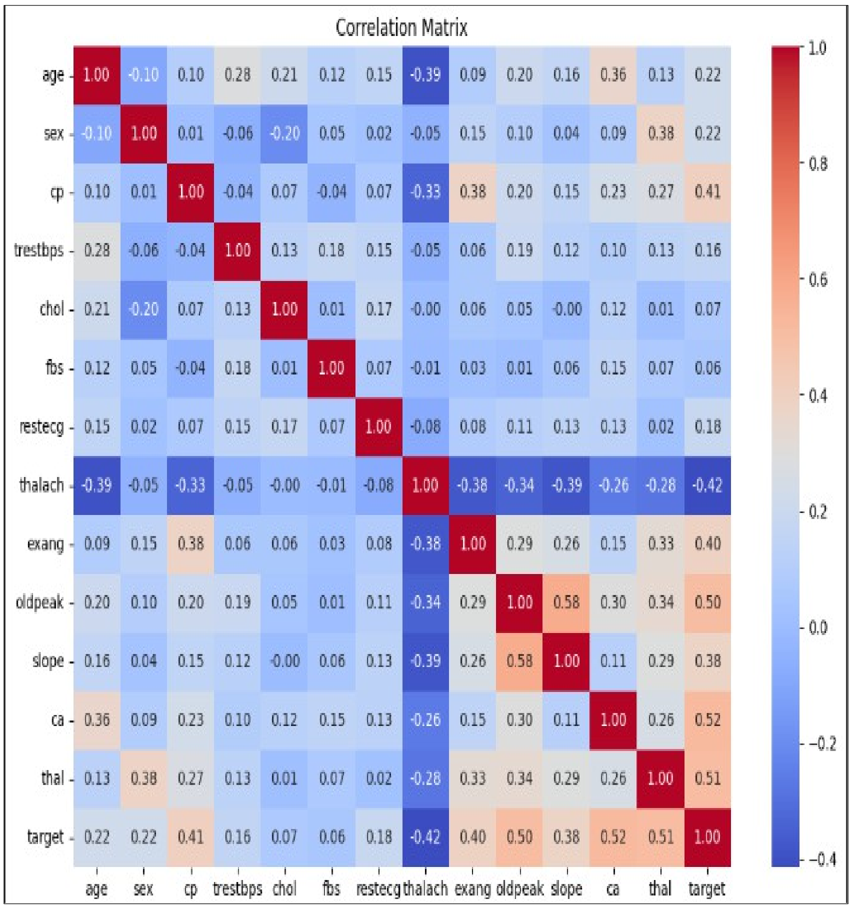


Fig 5.3 Correlation Matrix

5.4 ETHICAL CONSIDERATIONS IN AI

Ethical considerations are essential in AI healthcare applications, such as heart disease prediction using federated learning. Privacy and data security are prioritized, with federated learning helping to keep data decentralized, while encryption protects model updates. Data bias and fairness must be addressed to ensure the model is trained on diverse, representative data to avoid unfair outcomes. Informed consent ensures patients are aware of how their data is used and protected.

The model should be transparent and interpretable, using explainable AI techniques to build trust with healthcare providers. Accountability is critical, ensuring responsibility for incorrect predictions. Equity of access must be ensured, especially for underserved populations, to prevent widening healthcare disparities. Lastly, continuous monitoring and updates are necessary to maintain the model’s accuracy over time.

By addressing these ethical concerns, AI in healthcare can be used responsibly to improve patient outcomes while maintaining trust.

CHAPTER 6

SYSTEM REQUIREMENTS

6.1 Hardware Requirements

1. Central Server

- Processor: Quad-Core (e.g., Intel i7 or AMD Ryzen 5)
- RAM: 16 GB or higher
- Storage: 1 TB HDD or 512 GB SSD
- GPU: NVIDIA GTX 1060 or higher (for model aggregation and computation)
- Network: High-speed internet connection for secure communication with clients

2. Client Devices

- Processor: Dual-Core or higher
- RAM: 8 GB or higher
- Storage: 256 GB HDD or SSD
- GPU: Optional (required only for heavy local computations)
- Network: Stable internet connection for communication

6.2 Software Requirements

1. Operating System

- Windows 10/11, macOS, or Linux (Ubuntu 20.04 or later recommended)

2. Programming Languages

- Python 3.8 or higher

3. Frameworks and Libraries

- **TensorFlow**: For building and training the neural network models.
- **Keras**: For high-level neural network API integration.
- **Scikit-learn**: For evaluation metrics and model interpretation.
- **NumPy**: For numerical computations and data manipulation.
- **Pandas**: For managing and preprocessing datasets.
- **Cryptography Libraries**: For implementing secure key exchanges and encryption mechanisms (e.g., PyCryptodome).

4. **Additional Tools**

- Jupyter Notebook or an IDE (e.g., PyCharm, VS Code) for development.
- Docker (optional) for containerizing the federated learning environment.

6.3 Hardware Description

The hardware architecture for this project includes a central server and multiple client devices. The central server is responsible for aggregating model updates using the Federated Averaging Algorithm and securely managing client communications via cryptographic protocols. It requires significant computational resources, including high RAM and GPU power, to handle multiple client updates efficiently.

The client devices act as decentralized nodes that locally train their respective models using a neural network. They process datasets, update local model weights, and securely exchange them with the server. The hardware demands for clients are moderate, as most computations are lightweight and performed locally.

CHAPTER 7

SYSTEM TESTING

System testing ensures the entire heart disease prediction system operates seamlessly and meets all functional and non-functional requirements. It validates the integration of client-side training, secure communication, and centralized model aggregation, ensuring the system works as a cohesive unit. Functional tests check whether the system correctly predicts heart disease based on input features, while non-functional tests evaluate performance, scalability, and compliance with privacy standards. Scenarios such as network interruptions, varying client data distributions, and cryptographic protocol robustness are tested to ensure fault tolerance and security. The testing process also measures global model accuracy using metrics like precision and recall to confirm reliability. By simulating real-world conditions, system testing ensures the system's scalability, privacy preservation, and readiness for deployment in healthcare applications.

7.1 UNIT TESTING

Unit testing focuses on verifying the functionality of individual components of the system, such as the neural network model, cryptographic modules, and data preprocessing steps. Each module is tested independently to ensure that it performs as expected, such as accurate forward and backward propagation during model training or successful encryption and decryption of model updates. This testing isolates bugs early in development, ensuring the reliability of foundational components before integration.

7.2 INTEGRATION TESTING

Integration testing validates the seamless interaction between different components, such as the communication between client nodes and the central server, as well as the secure aggregation of model updates. It ensures that modules like local model training, cryptographic protocols, and federated averaging work cohesively without errors. The tests focus on data flow, synchronization, and the correctness of the global model update to guarantee smooth system operation in a distributed environment.

7.3 PERFORMANCE TESTING

Performance testing evaluates the system's efficiency, scalability, and resource utilization under various conditions. This includes testing the speed of model training and aggregation, the handling of multiple client connections, and the system's ability to scale across large datasets or numerous clients. Key performance metrics such as training time per round, server throughput, and client-server communication latency are analyzed to ensure optimal performance without compromising privacy or accuracy.

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

CONCLUSION

The heart disease prediction system using federated learning demonstrates the potential of decentralized AI to address critical healthcare challenges while ensuring data privacy. By training neural network models locally on client datasets and securely aggregating updates on a central server, the system achieves robust global performance without compromising patient confidentiality. The implementation of cryptographic protocols ensures the integrity and security of data exchanges, aligning with modern privacy standards like HIPAA and GDPR. The results highlight the model's ability to predict heart disease accurately and efficiently, paving the way for early diagnosis and timely intervention. This project showcases how federated learning can revolutionize healthcare by enabling collaborative AI solutions that prioritize privacy and security.

FUTURE ENHANCEMENTS

The next phase of this project focuses on expanding the dataset to include detailed information about specific heart conditions, enabling the model to predict not only the likelihood of heart disease but also the exact condition affecting a patient. By leveraging advanced clustering techniques and the knowledge from the current model, the neural network will autonomously identify patterns corresponding to different heart diseases. Additionally, incorporating techniques like transfer learning and explainable AI will improve the model's accuracy and interpretability. Integrating real-time data from wearable devices and IoT platforms can further enhance the system’s capability, making it adaptable to real-world, continuous monitoring scenarios. These advancements will make the system more comprehensive, precise, and impactful in healthcare applications.

APPENDICES

A.Sample Code

1. Setting Up the Server

```
server_ip = 'localhost'  # Server runs on the local machine
server_port = 5000 # Port number for client-server communication

server_socket=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((server_ip, server_port))  # Bind the socket to the address
server_socket.listen(3)  # Allow up to 3 clients to connect
```

Purpose:

Sets up the server to accept client connections using the socket library. The server listens on localhost and port 5000.

Key Points

1. socket.AF_INET: Indicates the use of IPv4 addressing.
2. socket.SOCK_STREAM: Specifies a TCP connection for reliable communication.
3. 3. server_socket.listen(3): Limits the server to handle 3 clients simultaneously.

2. Generating Cryptographic Keys

```
server_private_key = ec.generate_private_key(ec.SECP256R1()# Generate ECC private key
server_public_key = server_private_key.public_key()          # Derive ECC public key
```

Purpose

Uses Elliptic Curve Cryptography (ECC) to generate secure private and public keys for the server. These keys are used to securely exchange information with clients.

Key Points:

- **Private Key:** Known only to the server; used to compute a shared secret during key exchange.
- **Public Key:** Shared with clients; used to establish trust and secure communication.

3. Accepting Client Connections

```
client_connections = []

for i in range(3):  # Loop to connect with 3 clients
    conn, addr = server_socket.accept()  # Wait for a client to connect
    print(f"Client {i+1} connected from {addr}")
    client_connections.append(conn)  # Save the client's connection object

    # Send server's public key to the client
    conn.send(server_public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    ))
```

Purpose

Accepts connections from 3 clients and sends the server's public key to them for secure key exchange.

Key Points

- server_socket.accept(): Blocks until a client connects, then returns the connection object and the client’s address.
- The public key is sent to the client using PEM format for easy serialization.

4. Receiving Client Public Keys

```
client_public_keys = []

for i in range(3):
    client_public_key_pem = client_connections[i].recv(1024)  # Receive client's public key
    client_public_key    =    serialization.load_pem_public_key(client_public_key_pem)          #
Deserialize it
    client_public_keys.append(client_public_key)  # Store for secure communication
```

Purpose

Receives and stores each client's public key. These keys are needed to compute the shared secret for secure encryption/decryption.

Key Points

- recv(1024): Reads up to 1024 bytes of data (client's public key in PEM format).
- serialization.load_pem_public_key: Converts the PEM-formatted key back into an object usable by the cryptography library.

5. Building the Neural Network

```
def build_model():
    model = Sequential([
        Dense(16, activation='relu', input_shape=(13,)),  # Hidden layer 1
        Dense(8, activation='relu'),         # Hidden layer 2
        Dense(1, activation='sigmoid')       # Output layer for binary classification
    ])
    model.compile(optimizer=SGD(learning_rate=0.01),loss='binary_crossentropy',
metrics=['accuracy'])
    return model
```

Purpose

Defines a feed-forward neural network for binary classification (e.g., predicting heart disease).

Key Points

Layers

Input Layer: Expects 13 input features.

Two Hidden Layers: Use ReLU activation for non-linearity.

Output Layer: Uses a sigmoid activation for binary output (0 or 1).

Optimizer: Stochastic Gradient Descent (SGD) with a learning rate of 0.01.

Loss Function: Binary cross-entropy, suitable for binary classification problems.

6. Federated Learning Rounds

```
num_rounds = 40  # Total number of training rounds


for round_num in range(num_rounds):
   print(f"Starting round {round_num + 1}/{num_rounds}")


   client_weights = []  # To store weights received from all clients
```

Purpose

Orchestrates multiple federated learning rounds where the server updates a global model based on aggregated client weights.

6.1. Sending Global Weights to Clients

```
data = pickle.dumps(global_weights)  # Serialize global weights


shared_key = server_private_key.exchange(ec.ECDH(), client_public_keys[i])  # Generate shared key
derived_key = HKDF(algorithm=hashes.SHA256(), length=32, salt=None, info=b'handshake data').derive(shared_key)


aesgcm = AESGCM(derived_key)  # Initialize AES-GCM for encryption
nonce = b'\x00' * 12  # Static nonce for simplicity
encrypted_data = aesgcm.encrypt(nonce, data, None)  # Encrypt the global weights


client_connections[i].send(encrypted_data)  # Send encrypted weights to client
```

Purpose

Sends the global model's weights securely to each client using AES-GCM encryption.

6.2. Receiving Local Weights from Clients

```
encrypted_data = client_connections[i].recv(4096)  # Receive encrypted weights
data = aesgcm.decrypt(nonce, encrypted_data, None)  # Decrypt the received data
local_weights = pickle.loads(data)  # Deserialize local weights
client_weights.append(local_weights)  # Add to list of received weights
```

Purpose

Receives encrypted model weights from clients, decrypts them, and deserializes them for aggregation.

6.3. Aggregating Client Weights

```
new_weights = []  # Placeholder for updated global weights
for weights_list in zip(*client_weights):  # Group weights by layers
    new_weights.append(np.mean(weights_list, axis=0))  # Average weights layer-by-layer

global_weights = new_weights  # Update global model weights
model.set_weights(global_weights)  # Update the model
```

Purpose

Aggregates client weights using the Federated Averaging (FedAvg) algorithm by averaging layer-wise weights.

7. Saving the Trained Model

```
model.save('federated_model.h5')  # Save the final global model
```

Purpose

Saves the trained global model to a file for later use.

8. Closing Connections

```
for conn in client_connections:
    conn.close()  # Close each client connection

print("Federated learning complete and connections closed.")
```

Purpose

Ensures all client connections are properly closed after the federated learning process is complete.

Overall Flow of the Code

1. The server establishes secure communication with multiple clients.
2. It coordinates the training process by distributing global weights and receiving updated weights.

3. Weights are securely exchanged using ECC and AES-GCM encryption.
4. The server aggregates the received weights to improve the global model.
5. After completing all rounds, the global model is saved, and connections are closed.

Client Code

1. Setting Up the Client Connection

```
server_ip = 'localhost'
server_port = 5000
server_address = (server_ip, server_port)

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(server_address)  # Connect to the server
```

Purpose

This section establishes a TCP connection from the client to the server using the **socket** library. The client will connect to localhost at port 5000.

Key Points

- The client uses socket.AF_INET for IPv4 and socket.SOCK_STREAM for a TCP connection.
- client_socket.connect(server_address): The client attempts to connect to the server at the specified address.

2. Key Exchange Sending and Receiving Public Keys

```
server_public_key_pem = client_socket.recv(1024)  # Receive public key
server_public_key = serialization.load_pem_public_key(server_public_key_pem)  # Deserialize the key
```

Purpose

The client receives the server's public key to securely exchange data during communication.

- serialization.load_pem_public_key**:** Converts the received PEM public key into an object.

```
client_private_key = ec.generate_private_key(ec.SECP256R1())
client_public_key = client_private_key.public_key()
```

```
client_socket.send(client_public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
))
```

Purpose

The client generates its own ECC private and public key pair. The private key stays secret, while the public key is sent to the server for further key exchange.

Key Points

- ec.generate_private_key**:** Generates an elliptic curve private key using the SECP256R1 curve.
- The client sends its public key to the server after converting it into PEM format.

3. Loading and Training the Local Model

```
data = pd.read_csv('heart2ex.csv')  # Load dataset
X = data.drop('target', axis=1).values  # Features
y = data['target'].values  # Labels
```

Purpose

The client loads its local dataset (heart2ex.csv) and preprocesses it for training. The target variable is separated from the features.

Key Points

- pd.read_csv**:** Loads a CSV dataset into a Pandas DataFrame.
- drop('target', axis=1)**:** Removes the target column to keep the input features.
- data['target']**:** Selects the target variable for training.

4. Model Definition and Training

```
def build_model():
    model = Sequential([
        Dense(16, activation='relu', input_shape=(13,)),
        Dense(8, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=SGD(learning_rate=0.01),loss='binary_crossentropy',
metrics=['accuracy'])
    return model
```

Purpose

This defines the architecture of the local neural network model using Keras. It has:

- **16 neurons** in the first hidden layer.
- **8 neurons** in the second hidden layer.
- **1 output neuron** for binary classification (heart disease present or not).

```
model = build_model()
model.fit(X, y, epochs=5, batch_size=32, verbose=1)
```

Purpose

The client trains the model locally on its dataset for 5 epochs. The optimizer used is **SGD** with a learning rate of 0.01.

5. Encrypting and Sending Local Weights

```
data = pickle.dumps(local_weights)  # Serialize the model weights
```

Purpose

Serializes the model's weights (parameters) into a byte stream for transmission.

```
shared_key = client_private_key.exchange(ec.ECDH(), server_public_key)  # Secure shared key exchange
derived_key = HKDF(algorithm=hashes.SHA256(), length=32, salt=None, info=b'handshake data').derive(shared_key)
```

Purpose

- **ECDH Key Exchange:** Client and server exchange keys to compute a shared secret using their public keys.
- **HKDF:** Derives a **256-bit key** from the shared secret for encryption.

6. Receiving and Decrypting Global Weights

```
encrypted_data = client_socket.recv(4096)  # Receive encrypted weights from the server
decrypted

_data = aesgcm.decrypt(nonce, encrypted_data, None) # Decrypt using AES-GCM
global_weights = pickle.loads(decrypted_data) # Deserialize global weights
```

Purpose

After receiving the global model weights from the server, the client decrypts them using the same AES-GCM encryption algorithm.

7. Updating Local Model

```python
model.set_weights(global_weights)  # Update the local model with the global weights
```

Purpose

The client updates its local model with the global weights received from the server, combining the server's aggregated model with the local one.

8. Closing the Connection

```
client_socket.close()  # Close the connection
```

Purpose

The client closes the socket connection after the communication is complete.

Overall Workflow

1. The client establishes a connection to the server.
2. The client receives the server’s public key and sends its own.
3. The client loads and trains its local model.
4. The client securely encrypts and sends its model weights to the server.
5. The client receives and decrypts the aggregated global model weights.
6. The client updates its model with the global weights.
7. The client closes the connection.

Prediction Code

```
import numpy as np
from tensorflow.keras.models import load_model  # Used to load the saved model
import pandas as pd

# Load the trained global model (saved from federated learning)
model = load_model('federated_model.h5')  # The model is saved as 'federated_model.h5'

# New patient data for prediction (Example data)
new_patient_data = {
    'age': 55,
    'sex': 1,  # 1 = male, 0 = female
    'cp': 3,  # Chest pain type (1-4)
    'trestbps': 120,  # Resting blood pressure
    'chol': 200,  # Cholesterol level
    'fbs': 0,  # Fasting blood sugar (0/1)
    'restecg': 0,  # Resting electrocardiogram (0-2)
    'thalach': 150,  # Maximum heart rate
    'exang': 0,  # Exercise-induced angina (0/1)
    'oldpeak': 1.5,  # ST depression
    'slope': 2,  # ST segment slope
    'ca': 0,  # Number of major vessels colored
    'thal': 3  # Thalassemia (0-3)
}

# Convert the new patient data into a NumPy array
new_patient_array = np.array([
    new_patient_data['age'],
    new_patient_data['sex'],
    new_patient_data['cp'],
    new_patient_data['trestbps'],
    new_patient_data['chol'],
    new_patient_data['fbs'],
    new_patient_data['restecg'],
    new_patient_data['thalach'],
    new_patient_data['exang'],
    new_patient_data['oldpeak'],
    new_patient_data['slope'],
    new_patient_data['ca'],
    new_patient_data['thal']
])

# Reshape the array to match the input shape expected by the model (1, 13)
new_patient_array = new_patient_array.reshape(1, -1)  # This reshapes the data into a 2D array (1 row, 13 features)

# Use the model to make a prediction on the new patient's data
prediction = model.predict(new_patient_array)  # Returns the probability of heart disease

# Print the predicted probability and classification result
print("Heart Disease Probability:", prediction[0][0])

# Output the prediction result (whether heart disease is present or not)
if prediction[0][0] > 0.5:  # Threshold for classification
    print("Predicted Class: Heart Disease Present")
else:
    print("Predicted Class: Heart Disease Not Present")
```

Detailed Explanation of Each Section

1. Loading the Trained Model

```
model = load_model('federated_model.h5')  # The model is saved as 'federated_model.h5'
```

Purpose:

This line loads the globally trained model, which has been saved as a .h5 file (federated_model.h5).

Key Points:

- load_model() is a function from **Keras** that loads the entire model (architecture, weights, optimizer, etc.) for further inference.

2. New Patient Data Preparation

```
new_patient_data = {
    'age': 55,
    'sex': 1,  # 1 = male, 0 = female
    'cp': 3,  # Chest pain type (1-4)
    'trestbps': 120,  # Resting blood pressure
    'chol': 200,  # Cholesterol level
    'fbs': 0,  # Fasting blood sugar (0/1)
    'restecg': 0,  # Resting electrocardiogram (0-2)
    'thalach': 150,  # Maximum heart rate
    'exang': 0,  # Exercise-induced angina (0/1)
    'oldpeak': 1.5,  # ST depression
    'slope': 2,  # ST segment slope
    'ca': 0,  # Number of major vessels colored
    'thal': 3  # Thalassemia (0-3)
}
```

Purpose

A dictionary is created to represent the new patient’s data, where each key corresponds to a feature, such as age, sex, cholesterol, etc.

3. Data Conversion

```
new_patient_array = np.array([
    new_patient_data['age'],
    new_patient_data['sex'],
    new_patient_data['cp'],
    new_patient_data['trestbps'],
    new_patient_data['chol'],
    new_patient_data['fbs'],
    new_patient_data['restecg'],
    new_patient_data['thalach'],
    new_patient_data['exang'],
    new_patient_data['oldpeak'],
    new_patient_data['slope'],
    new_patient_data['ca'],
    new_patient_data['thal']
])
```

Purpose

This section converts the new_patient_data dictionary into a NumPy array for input to the model. The order of the features should match the order used during model training.

4. Reshaping the Data

```
new_patient_array = new_patient_array.reshape(1, -1)  # This reshapes the data into a 2D array (1 row, 13 features)
```

Purpose

The model expects the input data in a specific shape. Since the model expects a 2D array (with the first dimension representing the batch size), this line reshapes the data into a shape of (1, 13), where 1 is the number of patients (one sample) and 13 is the number of features.

5. Model Prediction

```
prediction = model.predict(new_patient_array)  # Returns the probability of heart disease
```

Purpose

The model makes a prediction using the new patient's data. The predict() function returns the **probability** that the patient has heart disease, as the model is trained for binary classification.

6. Displaying Prediction Results

```
print("Heart Disease Probability:", prediction[0][0])
```

Purpose

This prints the predicted probability of the patient having heart disease. The result is a value between 0 and 1, representing the model's confidence.

```
if prediction[0][0] > 0.5:  # Threshold for classification
    print("Predicted Class: Heart Disease Present")
else:
    print("Predicted Class: Heart Disease Not Present")
```

Purpose

A threshold of 0.5 is used to classify the result:

- If the probability is greater than 0.5, the model predicts that heart disease is present.
- If the probability is less than or equal to 0.5, the model predicts that heart disease is not present.

Overall Workflow

1. **Load the trained model** using load_model().
2. **Prepare the new patient's data** in a structured format (e.g., dictionary).
3. **Convert the patient data** into a NumPy array and reshape it to match the input shape required by the model.
4. **Make a prediction** using the model's predict() method.
5. **Output the prediction result**, including the probability and the class label (Heart Disease Present/Not Present).

Testing Code with Evaluation Metrics

```
import numpy as np
from tensorflow.keras.models import load_model  # Used to load the trained model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score
import pandas as pd

# Load the trained global model (saved from federated learning)
model = load_model('federated_model.h5')  # The model is saved as 'federated_model.h5'
# Load the test dataset (example: heart disease test data in CSV format)
# Assuming the test dataset has the same structure as the training data
test_data = pd.read_csv('heart2test.csv')

# Split the data into features (X) and labels (y)
X_test = test_data.drop('target', axis=1)  # 'target' is the column with heart disease labels (0 or 1)
y_test = test_data['target']
```

```
y_pred = model.predict(X_test)  # Returns probabilities for each instance
# Convert predicted probabilities to class labels (0 or 1) using a threshold of 0.5
y_pred_class = (y_pred > 0.5).astype(int)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred_class)  # Accuracy of the model
precision = precision_score(y_test, y_pred_class)  # Precision of the model
recall = recall_score(y_test, y_pred_class)  # Recall of the model
f1 = f1_score(y_test, y_pred_class)  # F1-score of the model
roc_auc = roc_auc_score(y_test, y_pred)  # ROC-AUC score

# Print evaluation results
print("Model Evaluation Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")
```

Detailed Explanation of the Testing Code

1. Loading the Trained Model

```
model = load_model('federated_model.h5')  # The model is saved as 'federated_model.h5'
```

Purpose:

This line loads the pre-trained global model (federated_model.h5), which was trained using federated learning on client data.

2. Loading the Test Dataset

```
test_data = pd.read_csv('heart2test.csv')
```

Purpose

The test data is loaded from a CSV file (heart2test.csv) containing features (e.g., age, cholesterol level, etc.) and the target labels (whether heart disease is present or not).

3. Splitting Features and Labels

```
X_test = test_data.drop('target', axis=1)  # 'target' is the column with heart disease labels (0 or 1)
y_test = test_data['target']
```

Purpose

- o X_test contains the features for the test set (input data, such as age, cholesterol, etc.).
- o y_test contains the true labels (target column), where 0 represents no heart disease and 1 represents heart disease.

4. Making Predictions

```
y_pred = model.predict(X_test)  # Returns probabilities for each instance
```

Purpose

The model makes predictions on the test set. It returns the probability of each sample having heart disease (a value between 0 and 1).

5. Converting Probabilities to Class Labels:

```
y_pred_class = (y_pred > 0.5).astype(int)
```

Purpose:

The predicted probabilities are converted into binary class labels:

1 (heart disease present) if the probability is greater than 0.5.

0 (heart disease not present) otherwise.

6. Calculating Evaluation Metrics:

Accuracy:

```
accuracy = accuracy_score(y_test, y_pred_class)
```

Purpose:

Measures the percentage of correct predictions (both true positives and true negatives) out of all predictions.

Precision:

```
precision = precision_score(y_test, y_pred_class)
```

Purpose:

Measures the proportion of true positives among all instances predicted as positive. Precision is important when the cost of false positives is high.

Recall:

```
recall = recall_score(y_test, y_pred_class)
```

Purpose

Measures the proportion of true positives among all actual positive instances. Recall is important when the cost of false negatives is high.

F1 Score

```
f1 = f1_score(y_test, y_pred_class)
```

Purpose

The harmonic mean of precision and recall. It balances both metrics and is useful when you need to account for both false positives and false negatives.

ROC-AUC Score:

```
roc_auc = roc_auc_score(y_test, y_pred)
```

Purpose

Measures the area under the Receiver Operating Characteristic (ROC) curve. The ROC-AUC score evaluates the model's ability to discriminate between positive and negative classes.

7. Displaying the Results

```
print("Model Evaluation Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")
```

Purpose

This prints the computed evaluation metrics (accuracy, precision, recall, F1-score, and ROC-AUC) to the console with a precision of 4 decimal places for better readability.

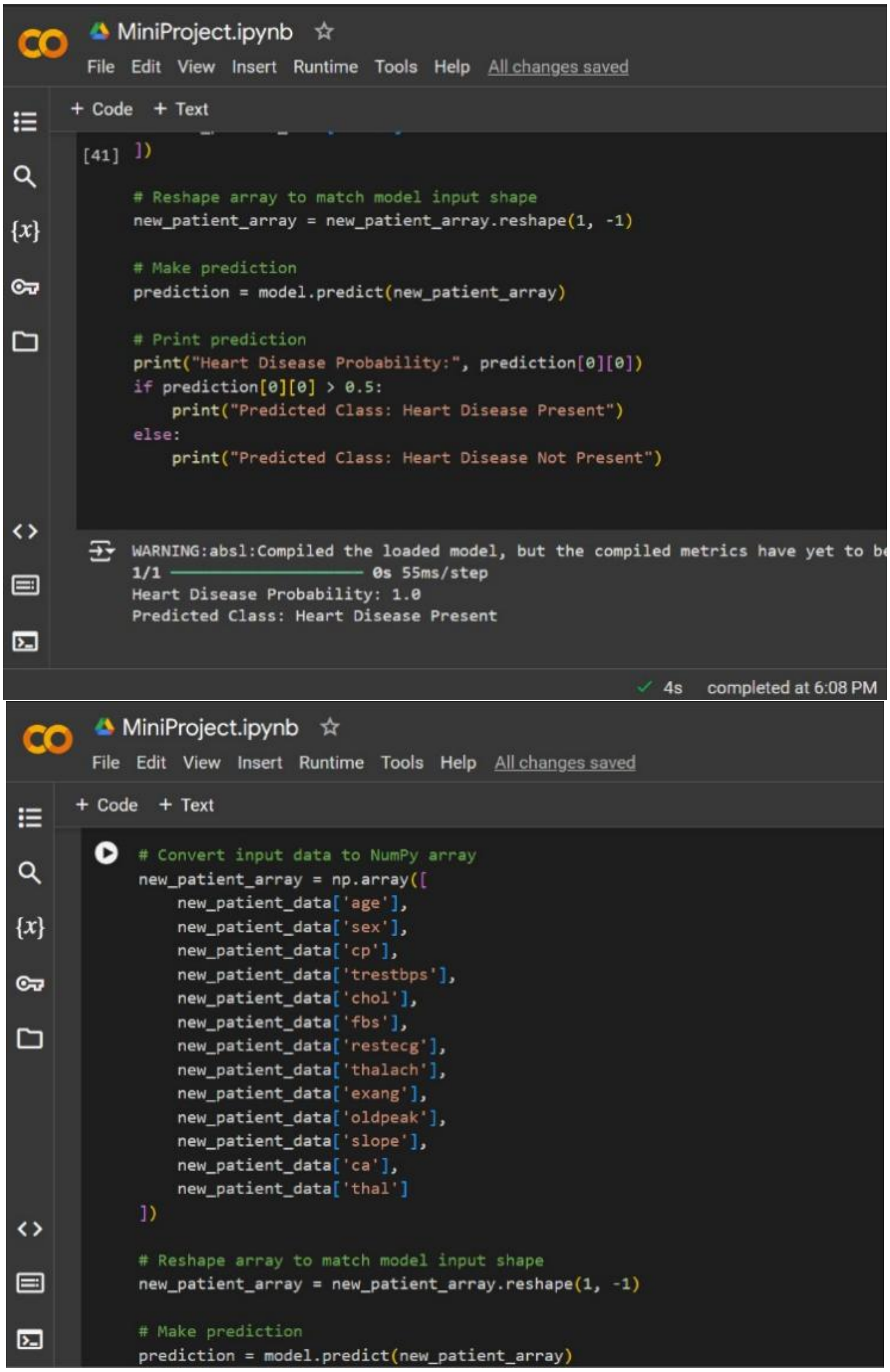
Overall Workflow

1. **Load the trained model** that was previously saved.
2. **Load the test dataset** containing patient data and heart disease labels.
3. **Split the dataset** into features (X) and labels (y).
4. **Make predictions** on the test data using the model.
5. **Convert predicted probabilities to binary class labels** using a threshold of 0.5.
6. **Calculate various evaluation metrics** (accuracy, precision, recall, F1-score, ROC-AUC).
7. **Print the results** of the evaluation metrics.

7. Conclusion of Software description

This federated learning system allows the development of privacy-preserving machine learning models by enabling secure and collaborative model training across multiple clients. By utilizing libraries like TensorFlow/Keras for model building, Cryptography for secure communication, and Pandas/NumPy for data handling, this software provides a robust, scalable, and secure framework for federated learning. With applications in various industries such as healthcare, finance, and IoT, the system ensures data privacy while enhancing the predictive power of machine learning models.

B. SCREENSHOT

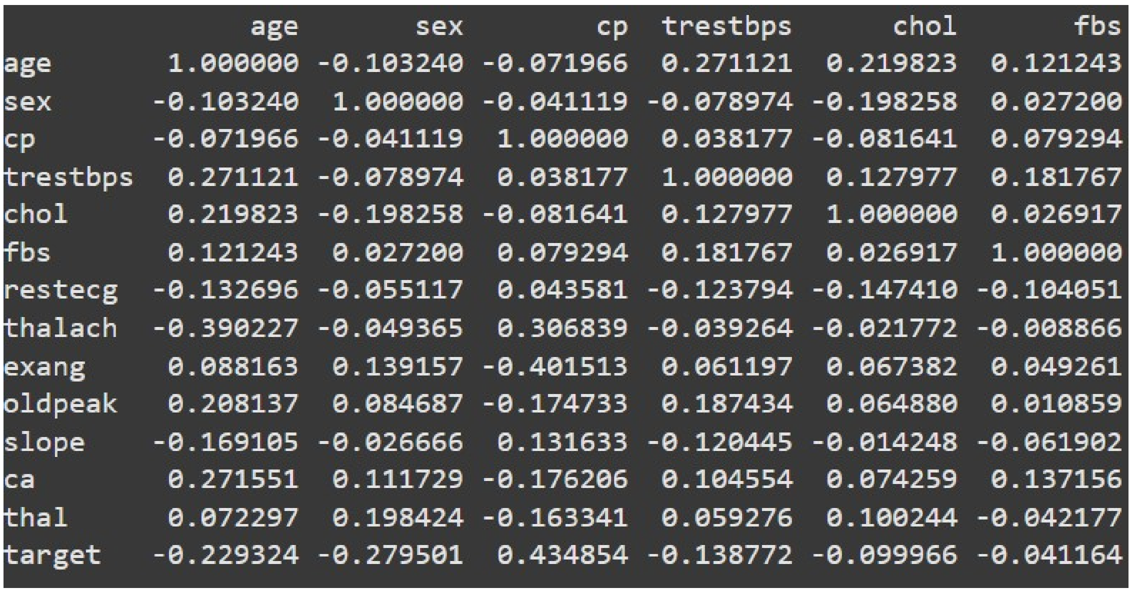

MiniProject.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
✓ 4s completed at 6:08 PM
MiniProject.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

```
[41] ])

    # Reshape array to match model input shape
    new_patient_array = new_patient_array.reshape(1, -1)

    # Make prediction
    prediction = model.predict(new_patient_array)

    # Print prediction
    print("Heart Disease Probability:", prediction[0][0])
    if prediction[0][0] > 0.5:
        print("Predicted Class: Heart Disease Present")
    else:
        print("Predicted Class: Heart Disease Not Present")
```

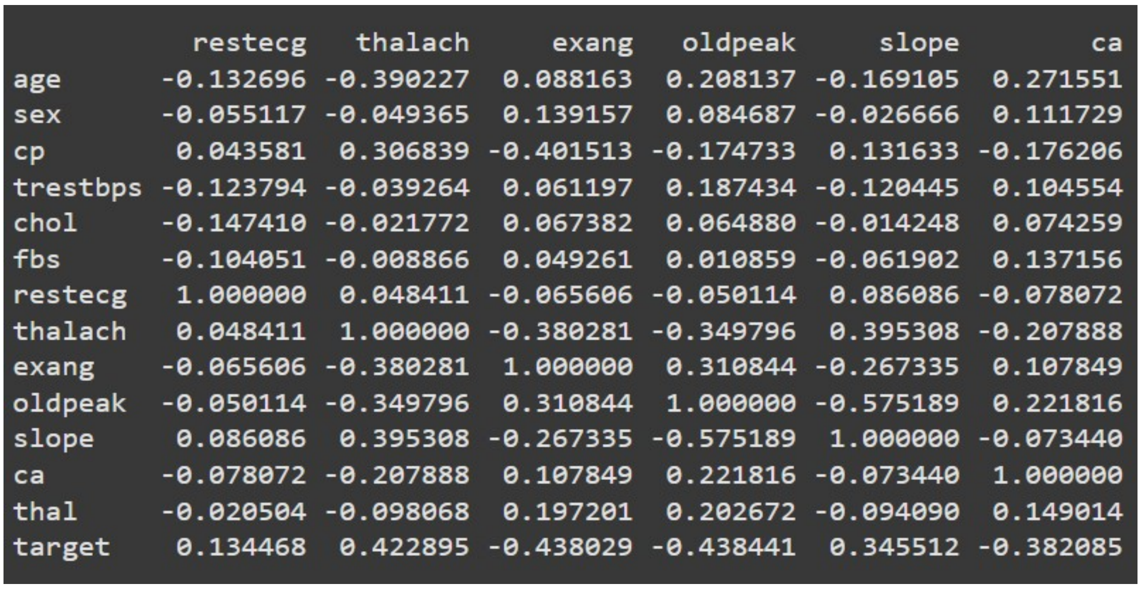
```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to b
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
Heart Disease Probability: 1.0
Predicted Class: Heart Disease Present
```

```
# Convert input data to NumPy array
new_patient_array = np.array([
    new_patient_data['age'],
    new_patient_data['sex'],
    new_patient_data['cp'],
    new_patient_data['trestbps'],
    new_patient_data['chol'],
    new_patient_data['fbs'],
    new_patient_data['restecg'],
    new_patient_data['thalach'],
    new_patient_data['exang'],
    new_patient_data['oldpeak'],
    new_patient_data['slope'],
    new_patient_data['ca'],
    new_patient_data['thal']
])

# Reshape array to match model input shape
new_patient_array = new_patient_array.reshape(1, -1)

# Make prediction
prediction = model.predict(new_patient_array)
```

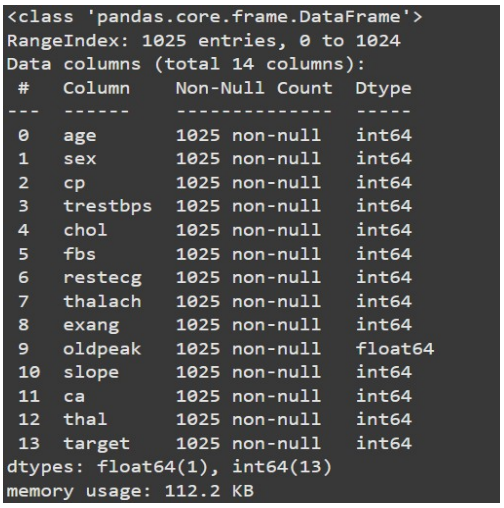
Fig B.1 Prediction Screen



```
               age       sex        cp  trestbps      chol       fbs
age       1.000000 -0.103240 -0.071966  0.271121  0.219823  0.121243
sex      -0.103240  1.000000 -0.041119 -0.078974 -0.198258  0.027200
cp       -0.071966 -0.041119  1.000000  0.038177 -0.081641  0.079294
trestbps  0.271121 -0.078974  0.038177  1.000000  0.127977  0.181767
chol      0.219823 -0.198258 -0.081641  0.127977  1.000000  0.026917
fbs       0.121243  0.027200  0.079294  0.181767  0.026917  1.000000
restecg  -0.132696 -0.055117  0.043581 -0.123794 -0.147410 -0.104051
thalach  -0.390227 -0.049365  0.306839 -0.039264 -0.021772 -0.008866
exang     0.088163  0.139157 -0.401513  0.061197  0.067382  0.049261
oldpeak   0.208137  0.084687 -0.174733  0.187434  0.064880  0.010859
slope    -0.169105 -0.026666  0.131633 -0.120445 -0.014248 -0.061902
ca        0.271551  0.111729 -0.176206  0.104554  0.074259  0.137156
thal      0.072297  0.198424 -0.163341  0.059276  0.100244 -0.042177
target   -0.229324 -0.279501  0.434854 -0.138772 -0.099966 -0.041164
```



```
           restecg   thalach     exang   oldpeak     slope        ca
age      -0.132696 -0.390227  0.088163  0.208137 -0.169105  0.271551
sex      -0.055117 -0.049365  0.139157  0.084687 -0.026666  0.111729
cp        0.043581  0.306839 -0.401513 -0.174733  0.131633 -0.176206
trestbps -0.123794 -0.039264  0.061197  0.187434 -0.120445  0.104554
chol     -0.147410 -0.021772  0.067382  0.064880 -0.014248  0.074259
fbs      -0.104051 -0.008866  0.049261  0.010859 -0.061902  0.137156
restecg   1.000000  0.048411 -0.065606 -0.050114  0.086086 -0.078072
thalach   0.048411  1.000000 -0.380281 -0.349796  0.395308 -0.207888
exang    -0.065606 -0.380281  1.000000  0.310844 -0.267335  0.107849
oldpeak  -0.050114 -0.349796  0.310844  1.000000 -0.575189  0.221816
slope     0.086086  0.395308 -0.267335 -0.575189  1.000000 -0.073440
ca       -0.078072 -0.207888  0.107849  0.221816 -0.073440  1.000000
thal     -0.020504 -0.098068  0.197201  0.202672 -0.094090  0.149014
target    0.134468  0.422895 -0.438029 -0.438441  0.345512 -0.382085
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

Fig B.2 Loaded data description

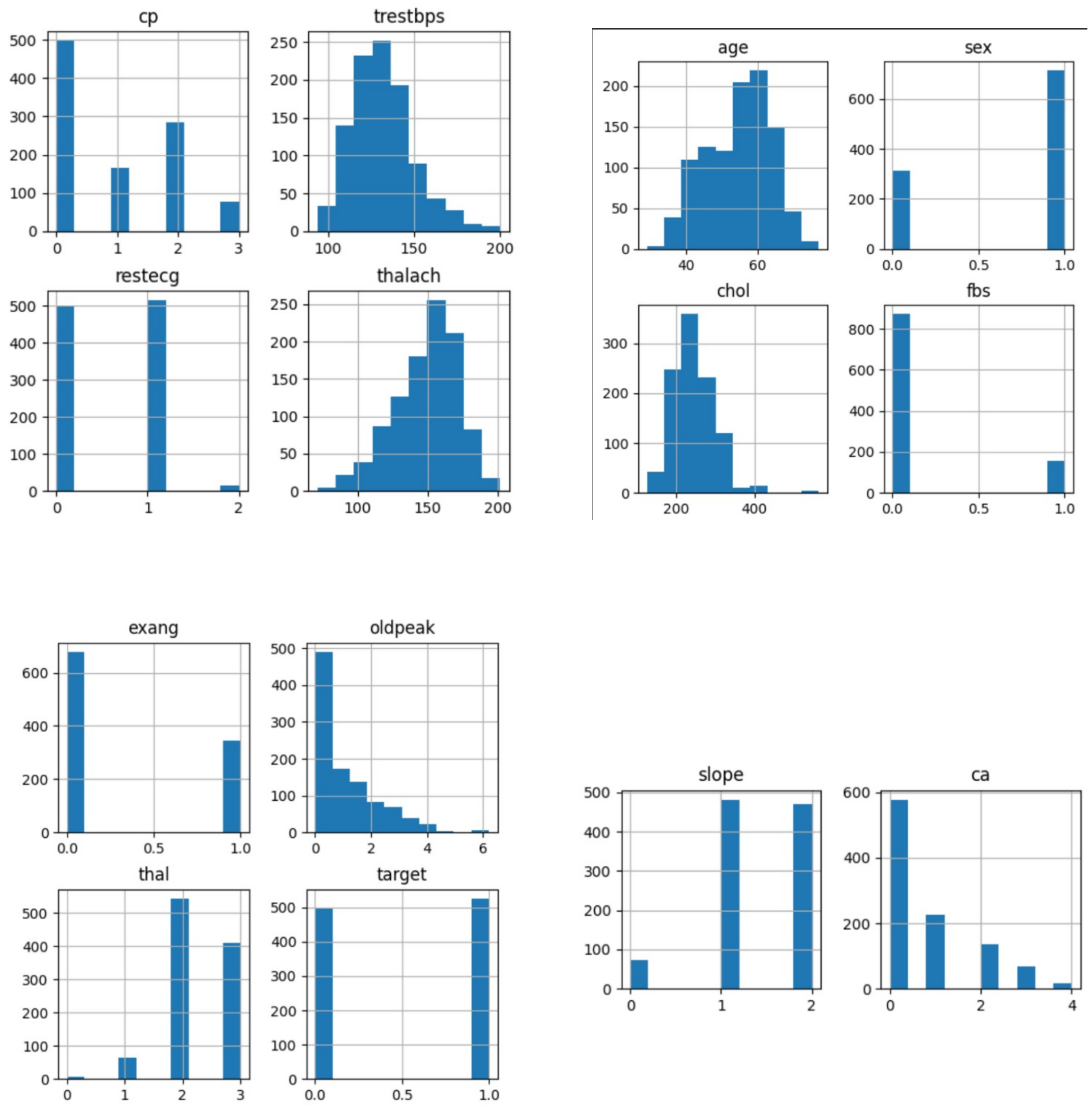


Fig B.3 Histogram Of Data

REFERENCES

1 Mohan, C. Thirumalai and G. Srivastava, "Effective Heart Disease Prediction Using Hybrid Machine Learning Techniques," in IEEE Access, vol. 7, pp. 81542-81554, 2019, doi: 10.1109/ACCESS.2019.2923707.

2 Abbaraju Sai Sathwik, Beebi Naseeba, Nagendra Panini Challa, "Cardiovascular Disease Prediction Using Hybrid-Random-Forest- Linear- Model (HRFLM)", 2023 IEEE World Conference on Applied Intelligence and Computing (AIC), pp.192-197, 2023.

3 A.Jenifer et al., "Edge-based Heart Disease Prediction Device using Internet of Things," 2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC), Salem, India, 2022, pp. 1500-1504, doi: 10.1109/ICAAIC53929.2022.9793104.

4 C. D. M and R. Senapati, "An Adoptive Heart Disease Prediction Model using Machine Learning Approach," 2022 OITS International Conference on Information Technology (OCIT), Bhubaneswar, India, 2022, pp. 49-54, doi: 10.1109/OCIT56763.2022.00020.

5 M. Jahir Pasha, Kiraniwale Aejaz Ahmed, Shaikh Mohammed Amair, Sunni Arshad Hussain, Shaik Fayaz, "Smart Diagnosis of Human Cardiovascular Disease Using Machine Learning and Parametric Optimization Techniques", 2023 International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS), pp.483-491, 2023.

6 G. Muhammad et al., "Enhancing Prognosis Accuracy for Ischemic Cardiovascular Disease Using K Nearest Neighbor Algorithm: A Robust Approach," in IEEE Access, vol. 11, pp. 97879-97895, 2023, doi: 10.1109/ACCESS.2023.3312046.

7 C. Chakraborty and A. Kishor, "Real-Time Cloud-Based Patient-Centric Monitoring Using Computational Health Systems," in IEEE

8 Transactions on Computational Social Systems, vol. 9, no. 6, pp. 1613-1623, Dec. 2022, doi: 10.1109/TCSS.2022.3170375.

9 S. Rao, S. Kulkarni, S. Mehta and P. Tawde, "Edge Computing-Based Heart Rate Monitoring System using RNN and LSTM," 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 2023, pp. 1-4, doi: 10.1109/ICCCNT56998.2023.10308242.

10 R. Nagavelli and C. V. Guru Rao, "Degree of Disease possibility (DDP): A mining based statistical measuring approach for disease prediction in health care data mining," International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014), Jaipur, India, 2014, pp. 1-6, doi: 10.1109/ICRAIE.2014.6909265.