PRIVACY-PRESERVING LIVER FIBROSIS DIAGNOSIS USING ENCRYPTED FEDERATED LEARNING AND DEEP LEARNING

A Main project thesis submitted in partial fulfillment of requirements for the award of degree for VIII semester

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING WITH DATA SCIENCE

By

B. SAADHANA (21131A4403)

B. RAMCHARANI (21131A4412)

J. LAKSHMI SOWJANYA (21131A4416)

S. SIRISHA (22135A4406)

Under the esteemed guidance of

Mr. B. AJAY RAM,
(Assistant Professor)
Department of Computer Science and Engineering

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING (AUTONOMOUS)

Approved by AICTE & Affiliated to Andhra University, Visakhapatnam from 2022-25

(Affiliated to JNTU-Kakinada up to 2021-22)

VISAKHAPATNAM

2024 – 2025

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING
(Autonomous)
Approved by AICTE & Affiliated to Andhra University, Visakhapatnam from 2022-23
(Affiliated to JNTUK, Kakinada upto 2021-22)
Accredited twice by NAAC with 'A' Grade with a CGPA of 3.47/4.00
Madhurawada, Visakhapatnam - 530048

CERTIFICATE

This is to certify that the main project entitled "**Privacy-Preserving Liver Fibrosis Diagnosis using Encrypted Federated Learning and Deep Learning"** being submitted by

B. SAADHANA (21131A4403)

B. RAMCHARANI (21131A4412)

J. LAKSHMI SOWJANYA (21131A4416)

S. SIRISHA (22135A4406)

in partial fulfillment for the award of the degree "Bachelor of Technology" in Computer Science and Engineering with Data Science to the Jawaharlal Nehru Technological University, Kakinada is a record of Bonafide work done under my guidance and supervision during VIII semester of the academic year 2024-2025.

The results embodied in this record have not been submitted to any other university or institution for the award of any Degree or Diploma.

GUIDE

Mr. B. Ajay Ram
(Assistant Professor)
Department of CSE
GVPCE(A)

HEAD OF THE DEPARTMENT

Dr. Y. Anuradha
Professor and
Associate Head of CSE(DS)
Department of CSE
GVPCE(A)

Signature of Examiner

DECLARATION

We hereby declare that this project entitled **"PRIVACY-PRESERVING LIVER FIBROSIS DIAGNOSIS USING ENCRYPTED FEDERATED LEARNING AND DEEP LEARNING"** is a Bonafide work done by us and submitted to Department of Computer Science and Engineering, G.V.P College of Engineering (Autonomous) Visakhapatnam, in partial fulfillment for the award of the degree of B. Tech is of our own and it is not submitted to any other university or has been published any time before.

PLACE: VISAKHAPATNAM

DATE:

B. SAADHANA (21131A4403)

B. RAMCHARANI (21131A4412)

J. L.SOWJANYA (21131A4416)

S. SIRISHA (22135A4406)

ACKNOWLEDGEMENT

We would like to express our deep sense of gratitude to our esteemed institute **Gayatri Vidya Parishad College of Engineering (Autonomous)**, which has provided us an opportunity to fulfill our cherished desire.

We express our sincere thanks to our principal **Dr. A. B. KOTESWARA RAO, Gayatri Vidya Parishad College of Engineering (Autonomous)** for his encouragement to us during this project, giving us a chance to explore and learn new technologies in the form of mini projects.

We express our deep sense of Gratitude to **Dr. Y. ANURADHA, Associate Professor, Associate Head of B. Tech CSE with Data Science, Gayatri Vidya Parishad College of Engineering (Autonomous)** for giving us an opportunity to do the project in college.

We express our profound gratitude and our deep indebtedness to our Guide **Mr. B. AJAY RAM,** whose valuable suggestions, guidance and comprehensive assessments helped us a lot in realizing our project.

We also thank our Project Coordinator, **Mr. B. Ajay Ram, Assistant Professor, Department of Computer Science and Engineering,** for the kind suggestions and guidance for the successful completion of our project work.

B. SAADHANA(21131A4403)

B. RAMCHARANI(21131A4412)

J.L.SOWJANYA(21131A4416)

S. SIRISHA(22135A4406)

ABSTRACT

The rise of Deep Learning (DL) has significantly transformed medical applications, enabling the development of reliable and scalable diagnostic and prognostic models. However, the adoption of DL in healthcare remains limited due to the large volume of data required for training and the challenges associated with centralized data collection. Privacy concerns, regulatory constraints, and data ownership issues further hinder multi-institutional collaboration in medical imaging. Federated Learning (FL) has emerged as a promising solution by allowing decentralized model training across multiple institutions without sharing raw patient data. By enabling secure collaboration while preserving privacy, FL addresses key barriers in medical AI deployment. In this study, we explore the feasibility of FL for liver fibrosis staging, demonstrating its potential to enhance diagnostic accuracy while ensuring data confidentiality. The findings highlight the effectiveness of privacy-preserving FL frameworks for real-world medical applications, paving the way for secure and scalable AI-driven healthcare solutions.

Keywords

Federated Learning, Privacy-Preserving AI, Medical Imaging, Liver Fibrosis Staging, Deep Learning, Healthcare AI, Multi-Institutional Collaboration, Decentralized Learning

INDEX

CHAPTER – 1

INTRODUCTION

Liver fibrosis is a progressive condition characterized by excessive accumulation of extracellular matrix proteins, leading to liver dysfunction and, in severe cases, cirrhosis. Accurate staging of liver fibrosis, typically classified from F0 to F4, is critical for early diagnosis, monitoring disease progression, and guiding treatment decisions. Medical imaging techniques, such as ultrasound and MRI, are widely used for non-invasive fibrosis assessment, but manual interpretation is often subjective and time-consuming. Deep Learning (DL)-based automated analysis offers a promising solution for accurate and efficient staging of liver fibrosis. However, training robust DL models requires large-scale, diverse, and well-annotated datasets, which is a major challenge due to privacy regulations and data-sharing restrictions in healthcare.

Federated Learning (FL) addresses these challenges by enabling multiple medical institutions to collaboratively train models without sharing raw patient data. In FL, each institution trains a local model on its dataset, and only model updates are sent to a central server for aggregation. This approach ensures patient privacy while allowing multi-institutional collaboration, leading to more generalized and robust models for medical image analysis. However, FL is still susceptible to privacy risks, such as model inversion attacks, where adversaries attempt to reconstruct original patient data from shared model updates.

To enhance privacy protection and improve the performance, this study employs Variational Autoencoders (VAEs), Convolutional Neural Networks (CNNs)**,** and Advanced Encryption Standard (AES) in a federated setting for the detection and staging of liver fibrosis from F0 to F4. VAEs help in learning latent representations, CNNs extract meaningful features from medical images and classify the image, and AES serve as an additional privacy-preserving mechanism by ensuring only essential features are transmitted. By integrating these techniques within FL, this work ensures robust fibrosis detection while preserving data privacy and regulatory compliance.

CHAPTER – 2

LITERATURE SURVEY

2.1 LITERATURE SURVEY:

Title	Author	Publisher and year	Methodology	Dataset	Result
A Machine Learning-Based Method for Detecting Liver Fibrosis	Miguel Suárez, Raquel Martínez, Ana María Torres, Antonio Ramón, Pilar Blasco, and Jorge Mateo	MDPI, 2023	Comparison with eXtreme Gradient Boosting (XGB), KNN, Decision Tree (DT), Bayesian Linear Discriminant Analysis (BLDA), Support Vector Machine (SVM), and Logistic Regression (LR).	Data was derived from a multicentre retrospective cross-sectional study conducted in 4 hospitals in Mexico between 2014 and 2020.	Accuracy of 93.16%
Federated-Learning-based Hierarchical Diagnosis of Liver Fibrosis	Yueying Zhou, Xinping Ren, Xiaoying Zheng, Yongxin Zhu, Kang Xu, Shijin Song, LiTian	IEEE, 2022	Federated learning(ResNet18 , ResNet34, DenseNet121 and VGG16)	STE images of around 250 patients with liver diseases from Shanghai Ruijin Hospital	Accuracy of 70%
Accurate liver disease prediction system using convolutional neural network	Jeyalakshmi, Rangaraj	Indian Journal of Science and Technology, 2021	MCNN-LDPS, MLPNN	Indian Liver patient Dataset (ILPD)	Accuracy of 90.75%
A novel model for predicting fatty liver disease by means of an artificial neural network	Chen, Yi-Shu, Dan Chen, Chao Shen, Ming Chen, Chao-Hui Jin, Cheng-Fu Xu, Chao-Hui Yu, and You-Ming Li	Gastroenterol Rep (Oxf), 2021	ANN	Collected from People who underwent health check-ups in 3 medicals from China	Accuracy of 82.10%
Prediction of Liver Disease using Rprop, SAG and CNN	Deepa H Belavigi, Veena G S, Divakar Harekal	IJITEE, 2019	Rprop, SAG, CNN	Indian Liver Patient Dataset (ILPD)	Accuracy of 92.09%

2.2 SUMMARY OF LITERATURE SURVEY:

The literature survey explores advancements in liver disease prediction, focusing on various machine learning and deep learning methodologies, datasets, and performance metrics. The reviewed studies highlight the effectiveness of different models in diagnosing liver fibrosis and related diseases, providing valuable insights into model comparisons and future research directions.

1. Machine Learning and Deep Learning Techniques:

Several studies have utilized machine learning models such as eXtreme Gradient Boosting (XGB), Support Vector Machines (SVM), Logistic Regression (LR), and deep learning models like Convolutional Neural Networks (CNN) and Artificial Neural Networks (ANN) for liver disease prediction.

Comparative analysis between traditional ML models and deep learning techniques suggests that deep learning models often achieve higher accuracy due to their ability to capture complex patterns within medical data.

2. Hybrid and Advanced Models:

The integration of multi-channel convolutional neural networks (MCNN-LDPS) and multilayer perceptron neural networks (MLPNN) in liver disease prediction has shown promising results, with accuracy reaching 90.75%.

Hybrid models incorporating different AI techniques, such as Rprop, Stochastic Average Gradient (SAG), and CNN, have been effective, achieving an accuracy of 92.09%.

3. Datasets and Experimental Findings:

The datasets used in these studies include the Indian Liver Patient Dataset (ILPD), Shanghai Ruijin Hospital's STE images, and multicenter retrospective datasets from hospitals in Mexico and China.

The highest accuracy was observed in the study using XGB, KNN, Decision Tree (DT), Bayesian Linear Discriminant Analysis (BLDA), SVM, and LR, which achieved an accuracy of 93.16%, indicating the effectiveness of ensemble learning methods.

4. Evaluation Metrics:

Common evaluation metrics include accuracy, precision, recall, and F1-score.

The models with accuracy above 90% demonstrate strong predictive capabilities, whereas models with lower accuracy highlight the challenges of generalization across different datasets.

5. Future Directions:

Future research should explore advanced federated learning techniques, improved model ensembling, and multi-modal data integration for higher accuracy and privacy protection.

More extensive validation across diverse population datasets is necessary to ensure model robustness and reliability.

2.3 OBJECTIVES:

1. Development of a Federated Learning Framework for Liver Fibrosis Detection: Implementation of a decentralized learning approach that enables multiple institutions to collaboratively train a deep learning model without sharing raw ultrasonic images, preserving patient privacy and data security.

2. Integrate Variational Autoencoders (VAEs) and Convolutional Neural Networks (CNNs): Utilize VAEs for feature extraction and CNNs for fibrosis classification in both local and global models, ensuring robust learning and accurate fibrosis stage detection.

3. Implementation of Secure Model Weight Transmission Using AES Encryption: Encrypt model weights before transmitting them from local institutions to the global model for aggregation, ensuring secure communication and protection against unauthorized access.

4. Ensure Model Robustness Across Multiple Institutions: Train and evaluate the federated model using diverse datasets from different sources to improve generalization and reduce biases in liver fibrosis detection.

5. Investigate the Impact of rounds and epochs on Model Performance: Experiment with a different number of rounds/epochs to optimize the federated learning process.

CHAPTER - 3

EXISTING SYSTEM

3.1 METHODOLOGY:

Architecture of Federated Learning:

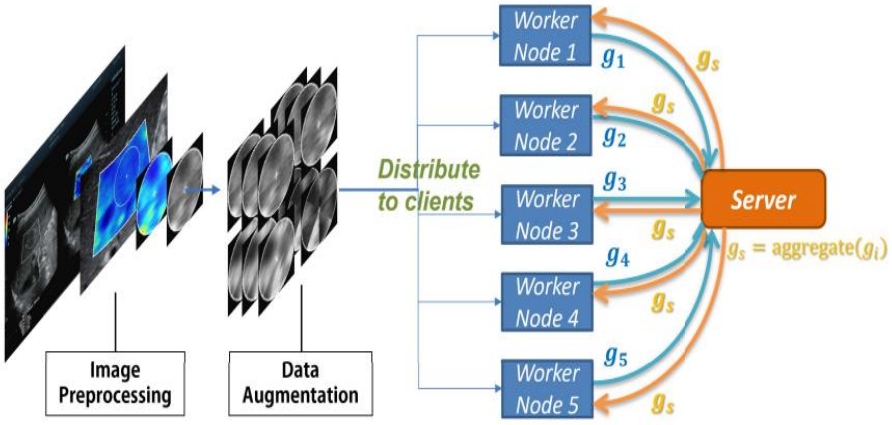

Fig 3.1- Federated System Structure

Clients (Local Devices / Nodes):

- These are the data owners (e.g., hospitals, mobile devices, or IoT sensors) that train local models using their private data.
- Each client updates its model parameters and sends them to the central server.

Central Server (Aggregator):

- The coordination point that collects local model updates from clients.
- It aggregates updates using techniques like Federated Averaging (FedAvg) or Adaptive Federated Averaging (AFA**)** to update the global model.

Local Models (Client-Side Models):

- Each client maintains its own machine learning model trained on local datasets.
- These models are periodically updated and sent to the central server.

Global Model:

- The central model that is trained using aggregated updates from multiple clients.
- It continuously improves as it receives **more local model updates** from different clients.

Working of Federated Learning:

- **Local Image Processing & Augmentation** – Each client node performs image augmentation (e.g., rotation, flipping, contrast adjustments) on its local dataset without sharing raw data.
- **Local Model Training** – Clients train a model using their augmented images and update model weights based on local computations.
- **Weight Exchange** – Instead of sending images, clients transmit only updated model weights to a central server.
- **Model Aggregation** – The central server aggregates the received weights (e.g., using Federated Averaging) to update the global model.
- **Iterative Learning** – The updated global model is sent back to clients, and the process repeats across multiple training rounds to improve accuracy while maintaining privacy.

EXISTING ALGORITHMS:

1. Machine Learning Algorithms Used in Federated Learning:

Logistic Regression (FL-LR) – Used for binary classification in federated environments.

Support Vector Machines (FL-SVM) – Applied in FL setups for classification tasks without requiring centralized data.

Random Forest (FL-RF) – Distributed decision trees for ensemble learning in federated networks.

Gradient Boosting Machines (FL-GBM) – Variants like XGBoost and LightGBM are adapted for FL by aggregating decision tree updates instead of raw data.

K-Means Clustering (FL-K-Means) – Used in FL for decentralized clustering of data across multiple nodes.

2. Deep Learning Algorithms Used in Federated Learning

FL has been integrated with various deep learning architectures, allowing large-scale distributed training.

Federated CNN (FL-CNN) – Convolutional Neural Networks adapted for FL, commonly used in image classification tasks.

3.2 PERFORMANCE MEASURES:

Accuracy – Measures the proportion of correct predictions out of total predictions.

Precision – The ratio of correctly predicted positive instances to total predicted positives (TP / (TP + FP)).

Recall (Sensitivity) – The ratio of correctly predicted positives to actual positives in the dataset (TP / (TP + FN)).

F1-Score – The harmonic mean of precision and recall, balancing both metrics (2 × (Precision × Recall) / (Precision + Recall)).

3.3 DRAWBACKS:

1. Traditional Methods

- **Operator Dependence** - Requires skilled operators; results vary between individuals.
- **Instrument Variability** - Different ultrasound machines give inconsistent results.
- **Limited Accuracy** - Biopsies are invasive; imaging methods have low sensitivity for early detection.

2. Machine Learning Methods

- **Data Privacy** - Centralized data collection raises security concerns (HIPAA, GDPR).
- **Limited Data Availability** - Difficult to collect large, diverse medical datasets.
- **Overfitting** - Models trained on small datasets may not generalize well.

3. Federated Learning (FL) Methods

- **Limited Dataset Availability** - Hospitals have small, isolated datasets.
- **No Ultrasonic Gray-Scale Images** - Most FL models rely on shear wave elastography (SWE) rather than grayscale ultrasound.
- **Heterogeneous Data** - Differences in imaging protocols create **non-IID** data issues.
- **Security & Compliance** - Needs **differential privacy, SMPC, and encryption** for secure model training.

CHAPTER - 4
PROPOSED SYSTEM

4.1 PROPOSED FEDERATED ARCHITECTURE

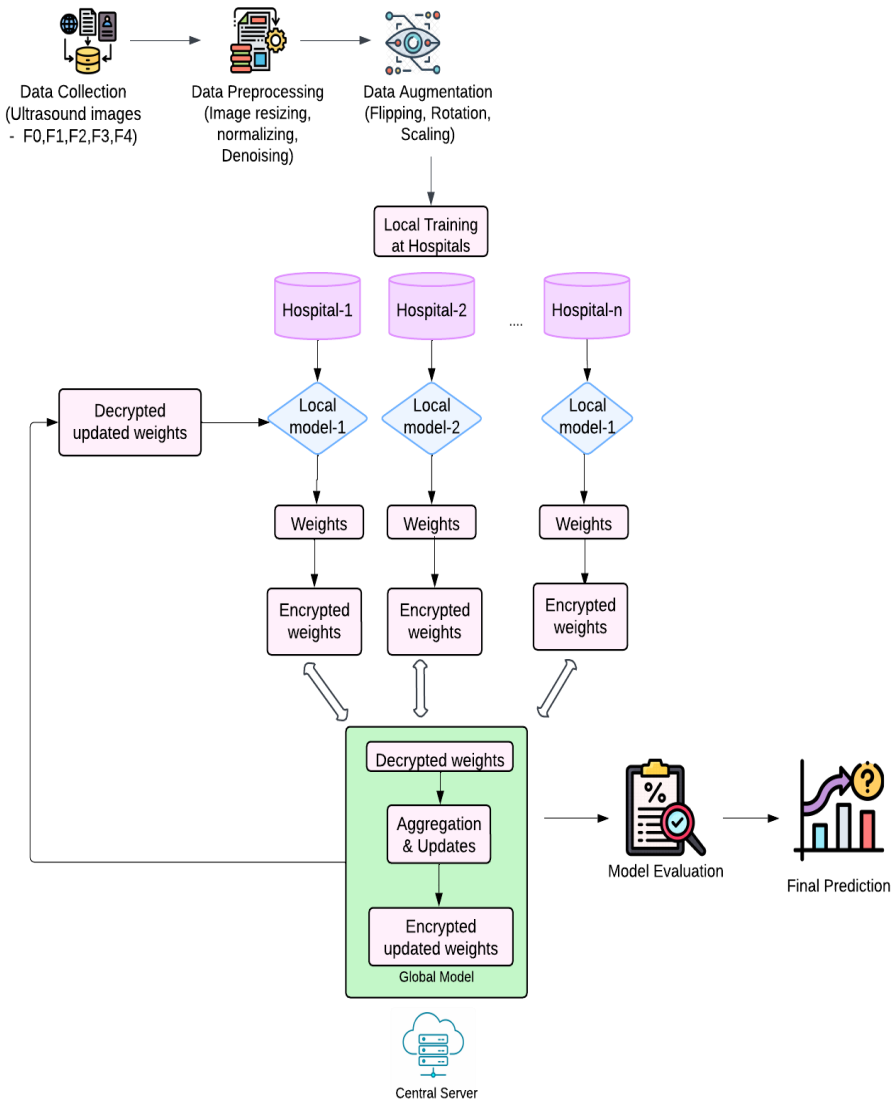

Fig 4.1 – Federated Architecture

1. Data Collection

The process begins with the collection of data from various sources, such as hospitals and medical institutions.

2. Data Preprocessing

The collected data undergoes preprocessing, which includes the following steps:

- Data cleaning to remove inconsistencies and missing values.
- Normalization to ensure uniformity in data representation.
- Feature extraction to identify relevant patterns in the data.

3. Feature Extraction

Relevant features are extracted from the preprocessed data to be used in training models. This step ensures that only meaningful information is utilized for diagnosis.

4. Local Training at Hospitals

Each participating hospital receives the extracted features and trains its own local model using its own data. The local models are trained independently at:

- Client-1 (Local Model-1)
- Client-2 (Local Model-2)
- Client-n (Local Model-n)

5. Weight Calculation

After training, each local model calculates its respective model weights, which represent the learning gained from the hospital's data.

6. Encryption of Weights

To ensure privacy and security, the calculated model weights are encrypted using Advanced Encryption Standards (AES) before being transmitted to the central server

7. Decryption and Aggregation of Weights

The encrypted weights from all hospitals are sent to a central server, where the following steps occur:

- Decryption: The encrypted weights are decrypted using AES.
- Aggregation: The weights are combined using federated learning techniques to create an improved global model.
- Encryption of Updated Weights: The newly updated global model weights are encrypted to maintain security.

8. Global Model Update

The encrypted updated global model weights are sent back to all participating hospitals to update their local models, ensuring improved accuracy and consistency.

9. Evaluation and Diagnosis

The updated global model is used for evaluating new patient data and diagnosing liver fibrosis. The evaluation process includes:

- Generating a diagnosis report with a checklist of symptoms.
- Calculating a prediction score to estimate fibrosis severity.
- Visualizing results through graphs and analytics.
- Providing insights for further medical investigations.

4.2 BLOCK DIAGRAM:

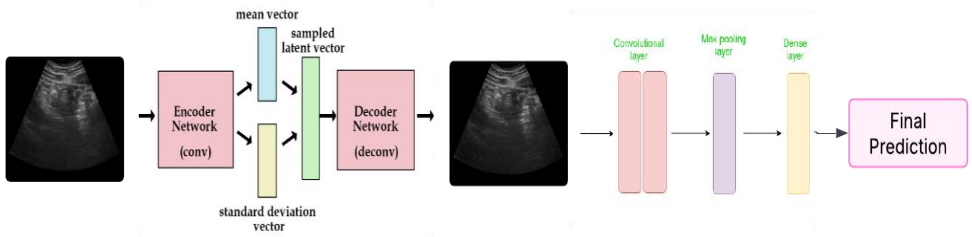

Fig 4.2 – VAE + CNN Block Diagram

The block diagram for the diagnosis of liver fibrosis using VAE+CNN outlines the key components and their interactions. Here's a brief description of each block:

Encoder Network – Compresses input data into a lower-dimensional latent space, producing a mean and variance for a probabilistic distribution.

Latent Space Sampling – Samples points from the latent distribution using the reparameterization trick to allow backpropagation.

Decoder Network – Reconstructs the input from the sampled latent representation.

Loss Function (Reconstruction + KL Divergence) – Optimizes the model by minimizing both reconstruction loss and KL divergence to ensure smooth latent space representation.

Generative Capability – Enables generation of new data by sampling from the learned latent space distribution.

Convolutional Layers – Extract hierarchical spatial features from input images using kernels/filters.

Activation Functions (ReLU) – Introduces non-linearity to enhance feature learning and prevent vanishing gradients.

Pooling Layers (Max/Average Pooling) – Reduce spatial dimensions while retaining essential features, improving computational efficiency.

Fully Connected Layers – Flatten and connect extracted features to dense layers for classification or regression tasks.

Softmax/Output Layer – Converts final layer outputs into probabilities for classification tasks.

4.3 MODULES:

1. tensorflow

- Used for building and training deep learning models.
- Provides layers, loss functions, and backend operations.

2. numpy

- Used for numerical computations and array operations.

3. cryptography.hazmat.primitives.ciphers

- Provides encryption and decryption functionalities.
- Uses AES encryption for secure model weight transmission.

4. cryptography.hazmat.backends

- Provides cryptographic backend support.

5. base64

- Used for encoding and decoding data for secure transmission.

6. os

- Used for file and directory operations.

7. random

- Generates random values, useful for data augmentation and encryption.

8. tensorflow.keras.preprocessing.image

- Provides tools for loading, processing, and augmenting images.

9. tensorflow.keras.layers, Model, Input

- Defines layers and architecture for neural networks.

10. tensorflow.keras.losses (MeanSquaredError, CategoricalCrossentropy)

- Loss functions for training and evaluating the model.

11. tensorflow.keras.backend (K)

- Provides low-level TensorFlow operations for custom functions.

12. tqdm

- Provides a progress bar for tracking the training process efficiently.

4.4 FLOW DIAGRAM:

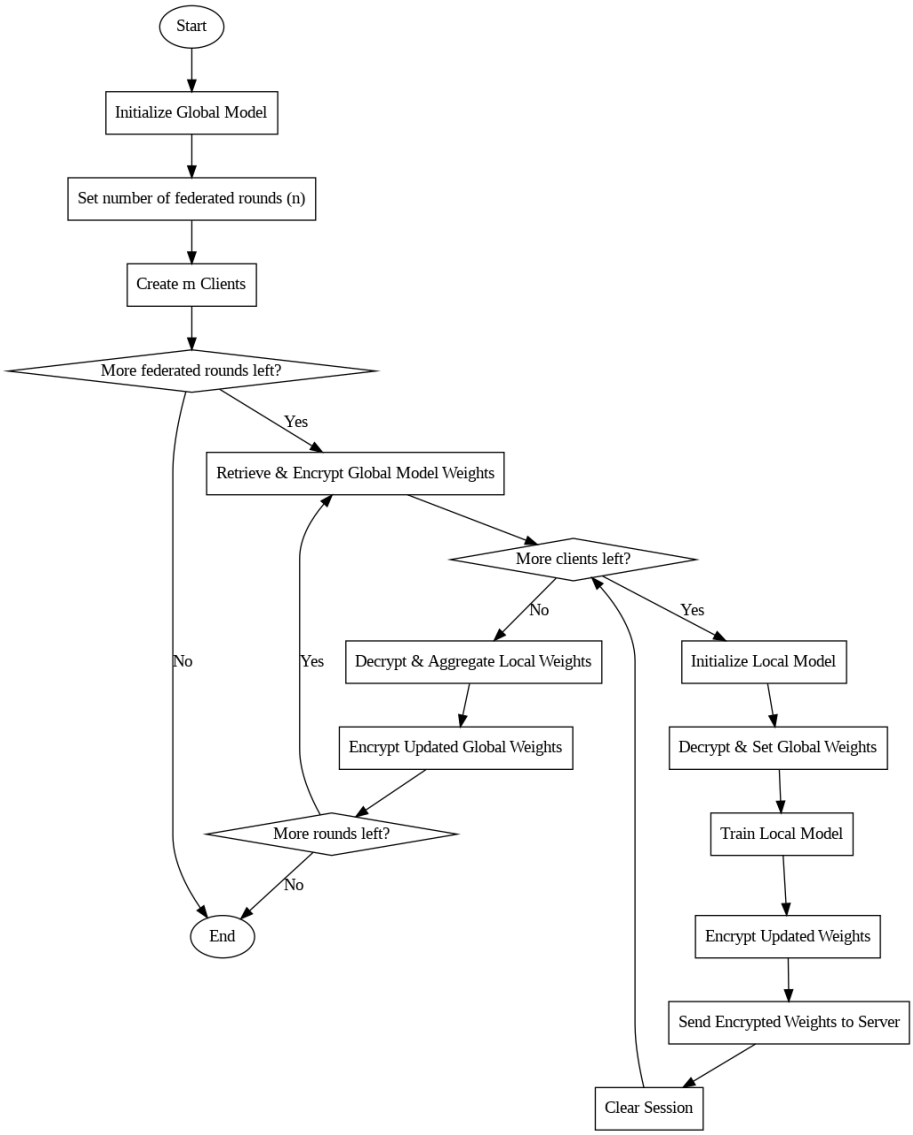

Fig 4.4 –Flow Diagram

4.5 METHODOLOGY:

1. Data Preparation & Preprocessing

- Collected multi-modality data:
  - Grayscale ultrasound images
- Preprocessed images using Gaussian filters for noise reduction.
- Applied data augmentation (flipping, rotation, scaling) to enhance model robustness.

- Converted labels into one-hot encoding for classification.

2. Federated Learning Setup

- Defined 9 client hospitals, each with its local dataset.
- Deployed a VAE-CNN model (Variational Autoencoder with CNN) at each client.
- Initialized a global model with random weights.
- Used AES and encrypted the weights to ensure privacy in weight updates.

3. Client-Side Model Training

- Distributed global model weights to all clients and then clients perform decryption of weights using AES.
- Each client:
  - Loaded and preprocessed images from their dataset.
  - Trained their local VAE-CNN model using Mean Squared Error (MSE) for autoencoder loss and Categorical Crossentropy for classification loss.
  - Encrypted trained model weights for security using AES.

4. Secure Weight Exchange & Aggregation

- Each client sent encrypted weights to the global server.
- The server decrypted and aggregated weights using Federated Averaging (FedAvg).
- The updated global model weights were sent back to all clients.

5. Convergence & Training Termination

- The process was iteratively repeated for up to 7 rounds.
- Stopped training if the loss improvement was below the threshold (0.0001 tolerance).

6. Global Model Testing & Evaluation

- The final global model was tested on a separate global test dataset.
- Performance Metrics Used:
  - Classification Accuracy
  - Categorical Cross Entropy Loss
  - Dice Coefficient (for segmentation & classification agreement)

7. Local Model Testing at Clients

- Evaluated client-side models using their respective test datasets.
- Compared local model performance with global model results.

4.6 PERFORMANCE MEASURE:

The performance of the Federated Learning (FL) model using VAE-CNN is evaluated using the following metrics:

1. **Classification Accuracy**: Measures the percentage of correctly classified fibrosis grades (F0–F4), indicating overall model performance.
2. **Categorical Cross Entropy Loss:** Computes the loss between the actual and predicted probability distributions, optimizing the classification task.
3. **Dice Coefficient:** Evaluates the overlap between predicted and actual fibrosis grade segmentations, crucial for medical image analysis.

Typically, well-trained **federated learning models** achieve an **accuracy of 85%–95%** for liver fibrosis diagnosis, ensuring reliable and privacy-preserving medical predictions.

4.7 ADVANTAGES:

1. **Privacy Preservation:** Patient data remains on local hospital servers, ensuring compliance with **HIPAA** and **GDPR** regulations while maintaining confidentiality.
2. **Decentralized Learning:** The model is trained across multiple hospitals without sharing raw data, enhancing security and preventing data breaches.
3. **Improved Generalization:** By learning from diverse datasets across different hospitals, the model becomes more robust and generalizes well across different patient populations.
4. **Scalability:** New hospitals can join the federated learning network without disrupting the existing training process, making it suitable for large-scale medical applications.
5. **Efficient Computation:** Local training reduces the need for centralized computing resources, making it feasible for hospitals with limited infrastructure.
6. **Early Diagnosis & Better Treatment Planning:** Accurate fibrosis grading (F0–F4) enables timely medical interventions, potentially preventing liver failure and improving patient outcomes.

CHAPTER – 5

DATASET AND MODEL / ALGORITHMS

5.1 DATASET

5.1.1 Web Reference Link

https://www.kaggle.com/code/houssameddinebhe/liver-histopathology-fibrosis-ultrasound-images

5.1.2 Metadata

Dataset Name: liver-histopathology-fibrosis-ultrasound-images

Dataset Description:

This dataset is derived from the research conducted by Y. Joo, H. -C. Park, O. -J. Lee, C. Yoon, M. H. Choi, and C. Choi, titled "Classification of Liver Fibrosis From Heterogeneous Ultrasound Image," published in IEEE Access, vol. 11, pp. 9920-9930, *2023*. The dataset includes ultrasound (US) images obtained from a tertiary university hospital (Seoul St. Mary's Hospital) used for training and validation purposes. Data from another university hospital (Eunpyeong St. Mary's Hospital) were utilized for testing.

Features:

F0 - No Fibrosis: 2114 images

Description: Represents healthy liver tissue with no signs of fibrosis. The tissue appears normal, with no scarring or fibrosis present.

F1 - Portal Fibrosis: 861 images

Description: Indicates fibrosis occurring around portal areas of the liver. Portal fibrosis involves the formation of scar tissue around the portal veins, which are small blood vessels in the liver.

F2 - Periportal Fibrosis: 793 images

Description: Shows fibrosis around the edges of the liver's portal areas. Periportal fibrosis is characterized by scarring around the liver's boundary regions, often affecting the areas near the portal veins.

F3 - Septal Fibrosis: 857 images

Description: Features fibrosis that forms bands or septa throughout the liver tissue. Septal fibrosis involves the development of thickened scar tissue that creates bands or partitions within the liver.

F4 - Cirrhosis: 1698 images

Description: Represents advanced fibrosis leading to cirrhosis. Cirrhosis is the most severe stage, marked by extensive scarring and loss of liver function, which can significantly impact liver health.

Data Format: Ultrasound Images

Size: 1.89GB

5.1.3 SAMPLE DATASET

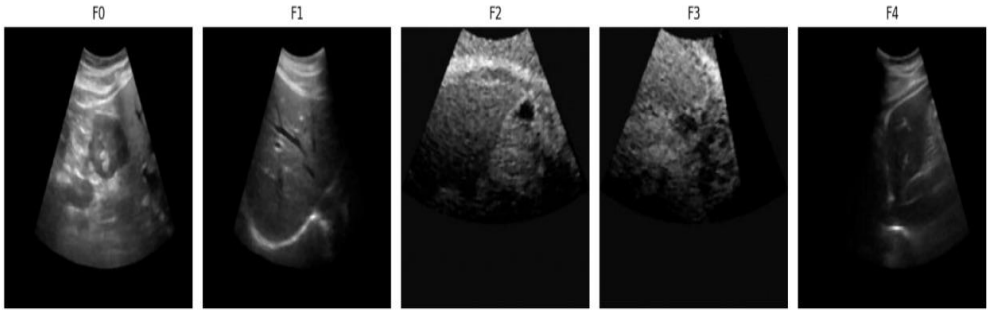

Fig 5.1.3 – Dataset Sample

5.2 MODELS/ALGORITHMS

Model Used: Variational Autoencoders(VAE) with Convolutional Neural Networks(CNN)

Framework: Keras(with Tensorflow backend)

Key Features of VAE and CNN:

VAEs extract meaningful features by encoding ultrasonic images into a compressed latent space, reducing noise and improving generalization. CNNs classify liver fibrosis by detecting spatial patterns and hierarchical features, enabling accurate differentiation between fibrosis stages.

Model Architecture:

1. **Input Layer** – Ultrasonic liver images are fed into the model.
2. **Encoder (VAE)** – Extracts latent features using convolutional layers.
3. **Latent Space (VAE)** – Encodes the extracted features into a lower-dimensional representation.

4. **Decoder (VAE)** – Reconstructs the input from latent features for regularization.
5. **CNN Classifier** – Uses extracted features to classify liver fibrosis stages.
6. **Output Layer** – Predicts the fibrosis stage (F1–F4).
7. **Federated Aggregation** – Local model weights are encrypted, sent to the global model, aggregated, and redistributed.

5.2.1 PSEUDO CODE

1. **Initialize** the global model.
2. **Set** the number of federated rounds **n**.
3. Create **m clients**.
4. **For each federated round** (up to **n**):
5. Retrieve the **global model's weights**.
6. **Encrypt** the global model's weights.
7. For each client (up to m):
8. Initialize the **local model**.
9. **Decrypt** the global model weights and set them as the local model's weights.
10. Train the **local model** using the client's dataset.
11. **Encrypt** the updated local model's weights.
12. Send the **encrypted local weights** to the global server.
13. Clear the **session**.
14. **Decrypt** all received local model weights at the global server.
15. Aggregate the **decrypted weights** from all local models.
16. **Encrypt** the updated global model's weights.
17. Send the **encrypted updated weights** back to clients.
18. **Repeat** for **n rounds**.

CHAPTER - 6

SAMPLE CODE

1. augmentation.py: In this file, the dataset was expanded from 6,323 to 21,120 images through augmentation to improve model robustness and performance.

```
import os
import random
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array, array_to_img
from tqdm import tqdm

# Define proper augmentation techniques
datagen = ImageDataGenerator(
    rotation_range=30,       # Rotate images by up to 30 degrees
    width_shift_range=0.2,   # Shift image horizontally
    height_shift_range=0.2,  # Shift image vertically
    shear_range=0.2,
    zoom_range=0.2,          # Random zoom
    horizontal_flip=True,    # Flip images horizontally
    fill_mode='nearest'      # Fill in missing pixels
)

# Function to balance dataset
def augment_images(data_path):
    clients = [os.path.join(data_path, d) for d in os.listdir(data_path)
               if os.path.isdir(os.path.join(data_path, d)) and "test" not in d.lower()]  # Exclude test da

    for client in clients:
        print(f"Processing {client}...")
        categories = [f'f{i}-{client[-1]}' for i in range(5)]  # Generate category names
        category_paths = {cat: os.path.join(client, cat) for cat in categories}
```

```
# Get max number of images in any category
max_images = max(len(os.listdir(path)) for path in category_paths.values() if os.path.exists(path))

for category, path in category_paths.items():
    if not os.path.exists(path):
        print(f"Warning: {path} does not exist. Skipping...")
        continue

    image_paths = [os.path.join(path, img) for img in os.listdir(path) if img.endswith(('png', 'jpg' 'jpeg'))
    num_images = len(image_paths)

    if num_images < max_images:
        print(f"Augmenting {category}: {num_images} → {max_images}")
        existing_images = [load_img(img_path, target_size=(128, 128)) for img_path in image_paths]

        for i in tqdm(range(max_images - num_images)):
            img = random.choice(existing_images)  # Select a random image
            img_array = img_to_array(img) / 255.0  # Normalize

            # Generate a random transformation
            transform_params = datagen.get_random_transform(img_array.shape)
            aug_img_array = datagen.apply_transform(img_array, transform_params)

            # Convert back to image
            aug_img = array_to_img(aug_img_array)
```

```
                # Save new image
                new_filename = os.path.join(path, f"aug_{i}.jpg")
                aug_img.save(new_filename)

        print(f"Finished processing {client}!\n")

# Run script
if __name__ == "__main__":
    dataset_path = "dataset"  # Update this if needed
    augment_images(dataset_path)
```

2. preprocessing.py: This file loads, preprocesses, and labels images from the dataset, ensuring they are resized, normalized, and converted into a format suitable for model training.

```
import tensorflow as tf
import numpy as np
import os
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.model_selection import train_test_split

def load_and_preprocess_images(data_path,categories):
    images = []
    labels = []
    # Update categories to match client folder structures

    for category in categories:
        category_path = os.path.join(data_path, category)
        label = categories.index(category)

        if not os.path.exists(category_path):
            print(f"Warning: Category folder {category_path} does not exist.")
            continue  # Skip if category folder does not exist

        for img_name in os.listdir(category_path):
            img_path = os.path.join(category_path, img_name)
            try:
                img = load_img(img_path, target_size=(128, 128))
                img_array = img_to_array(img) / 255.0
                images.append(img_array)
                labels.append(label)
            except Exception as e:
                print(f"Error loading image {img_path}: {e}")
```

```
    if not images:
        print(f"No images found in the path: {data_path}")
    else:
        print(f"Loaded {len(images)} images from {data_path}")

    images = np.array(images)
    labels = tf.keras.utils.to_categorical(labels, num_classes=5)

    if len(images) == 0 or len(labels) == 0:
        raise ValueError(f"Dataset at {data_path} is empty. Ensure the dataset contains valid images.")

    return images, labels
```

3. Client_training.py: client_training.py is responsible for training the local model on client data.

```python
import tensorflow as tf
import numpy as np
import os
from sklearn.model_selection import train_test_split

from encryption_decryption import encrypt_weights, decrypt_weights, reconstruct_weights
from preprocessing import load_and_preprocess_images
from client_testing import client_testing

client_test_data = {}

def train_model(model, client_data_path, epochs=4, loss_weights=[1.0, 0.5]):
    global client_test_data

    # Extract client identifier from the dataset path
    client_id = os.path.basename(client_data_path)
    categories = [f'f{i}-{client_data_path[-1]}' for i in range(5)]

    print(f"Loading data for client: {client_id}...")

    try:
        images, labels = load_and_preprocess_images(client_data_path, categories)
        if len(images) < 10:  # Ensure dataset is large enough
            print(f"Warning: Not enough images in {client_data_path} for training.")
            return None
    except Exception as e:
        print(f"Error loading images: {e}")
        return None
```

```python
    # Split dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)
    client_test_data[client_data_path[-1]] = (X_test, y_test)

    print(f"Training model for client: {client_id} with {len(X_train)} training images and {len(X_test)} te

    # Compile the model
    model.compile(
        optimizer='adam',
        loss={'decoder_output': 'mse', 'classifier_output': 'categorical_crossentropy'},
        loss_weights=loss_weights,
        metrics={'classifier_output': 'accuracy'}
    )

    # Train the model
    model.fit(
        X_train, [X_train, y_train],
        validation_data=(X_test, [X_test, y_test]),
        epochs=epochs
    )

    print(f"Training completed for client: {client_id}")

    return model.get_weights()  # Returning trained weights
```

4. encryption_decryption.py: This file is responsible for the encryption, decryption and reconstruction of the weights.

```
import numpy as np
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
import base64

SECRET_KEY = b'my_secret_key_16'  # Must be exactly 16, 24, or 32 bytes long

# Encrypt weights
def encrypt_weights(weights):
    serialized_weights = np.concatenate([w.flatten() for w in weights])  # Flatten all weights
    padded_length = (len(serialized_weights) + 15) // 16 * 16  # Ensure it's a multiple of 16
    padded_weights = np.pad(serialized_weights, (0, padded_length - len(serialized_weights)), 'constant')

    cipher = Cipher(algorithms.AES(SECRET_KEY), modes.ECB(), backend=default_backend())
    encryptor = cipher.encryptor()
    encrypted_weights = encryptor.update(padded_weights.astype(np.float32).tobytes()) + encryptor.finalize()

    return base64.b64encode(encrypted_weights).decode('utf-8')

# Decrypt weights
def decrypt_weights(encrypted_weights):
    encrypted_weights = base64.b64decode(encrypted_weights)  # Convert string back to bytes

    cipher = Cipher(algorithms.AES(SECRET_KEY), modes.ECB(), backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted_weights = decryptor.update(encrypted_weights) + decryptor.finalize()

    # Convert decrypted bytes back to NumPy array
    decrypted_array = np.frombuffer(decrypted_weights, dtype=np.float32)
```

```
    return decrypted_array  # Ensure this is a NumPy array

# Reconstruct weights
def reconstruct_weights(decrypted_weights, original_shapes):
    weights = []
    start = 0
    for shape in original_shapes:
        size = np.prod(shape)
        weights.append(decrypted_weights[start:start + size].reshape(shape))  # ✅ Now this won't fail
        start += size
    return weights
```

5. models.py: models.py defines the architecture of the Variational Autoencoder (VAE) and Convolutional Neural Network (CNN) used for training.

```
import tensorflow as tf
from tensorflow.keras import layers, Model, Input
from tensorflow.keras.losses import MeanSquaredError, CategoricalCrossentropy
import tensorflow.keras.backend as K
```

```
@tf.keras.utils.register_keras_serializable()
def sampling(args):
    """Reparameterization trick to sample latent vector z."""
    z_mean, z_log_var = args
    batch = K.shape(z_mean)[0]
    dim = K.int_shape(z_mean)[1]
    epsilon = K.random_normal(shape=(batch, dim))
    return z_mean + K.exp(0.5 * z_log_var) * epsilon

def create_vae_cnn_model(input_shape=(128, 128, 3), num_classes=5, latent_dim=64):
    """Creates a Variational Autoencoder (VAE) with CNN-based encoder, decoder, and classifier."""

    ### 🔹 Encoder ###
    inputs = Input(shape=input_shape, name="input_layer")

    x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 2))(x)
```

```
    x = layers.Flatten()(x)
    z_mean = layers.Dense(latent_dim, name="z_mean")(x)
    z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)

    # Sample z using reparameterization trick
    # Instead of using the function directly, reference it by name (optional but ensures consistency)
    z = layers.Lambda(sampling, output_shape=(latent_dim,), name="latent_space")([z_mean, z_log_var])

    ### 🔹 Decoder (Reconstructs Input) ###
    x = layers.Dense((input_shape[0] // 8) * (input_shape[1] // 8) * 128, activation='relu')(z)
    x = layers.Reshape((input_shape[0] // 8, input_shape[1] // 8, 128))(x)
    x = layers.Conv2DTranspose(128, (3, 3), activation='relu', padding='same')(x)
    x = layers.UpSampling2D((2, 2))(x)
    x = layers.Conv2DTranspose(64, (3, 3), activation='relu', padding='same')(x)
    x = layers.UpSampling2D((2, 2))(x)
    x = layers.Conv2DTranspose(32, (3, 3), activation='relu', padding='same')(x)
    x = layers.UpSampling2D((2, 2))(x)
    decoder_output = layers.Conv2DTranspose(input_shape[2], (3, 3), activation='sigmoid', padding='same', nar

    ### 🔹 Classifier (Predicts Class) ###
    classifier_output = layers.Dense(num_classes, activation='softmax', name="classifier_output")(z)

    ### 🔹 VAE Model ###
    model = Model(inputs=inputs, outputs=[decoder_output, classifier_output], name="VAE_CNN_Classifier")
```

```
    ### 🔷 Classifier (Predicts Class) ###
    classifier_output = layers.Dense(num_classes, activation='softmax', name="classifier_output")(z)

    ### 🔷 VAE Model ###
    model = Model(inputs=inputs, outputs=[decoder_output, classifier_output], name="VAE_CNN_Classifier")

    ### 🔷 Custom Loss (Reconstruction + KL Divergence) ###
    def vae_loss(y_true, y_pred):
        recon_loss = MeanSquaredError()(y_true, y_pred)
        kl_loss = -0.5 * K.sum(1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axis=-1)
        return recon_loss + 0.0001 * kl_loss  # Small weight for KL loss

    ### 🔷 Compile Model ###
    model.compile(
        optimizer='adam',
        loss={
            "decoder_output": vae_loss,  # Custom VAE Loss
            "classifier_output": CategoricalCrossentropy()  # Classification Loss
        },
        loss_weights={"decoder_output": 1.0, "classifier_output": 1.0},  # Equal importance
        metrics={"classifier_output": "accuracy"}
    )

    return model

# Create the model
vae_model = create_vae_cnn_model()
vae_model.summary()
```

6. weights_aggregation.py: This file aggregates model weights from multiple clients by averaging them, ensuring consistency, handling encryption decryption, and validating weight integrity.

```
import numpy as np
from encryption_decryption import decrypt_weights

def aggregate_weights(client_weights):
    """
    Aggregates model weights from multiple clients by averaging corresponding layers.

    :param client_weights: List of lists containing model weights from each client.
    :return: Aggregated global weights.
    """
    if not client_weights:
        print("Error: No client weights provided for aggregation.")
        return None

    print("Starting weight aggregation...")

    # Ensure all clients provided valid weights
    client_weights = [w for w in client_weights if w is not None]

    if len(client_weights) == 0:
        print("Error: No valid client weights to aggregate.")
        return None

    # Decrypt weights if they are encrypted
    client_weights = [decrypt_weights(w) if isinstance(w, bytes) else w for w in client_weights]

    # Ensure all clients have the same number of layers
    num_layers = len(client_weights[0])
    if any(len(weights) != num_layers for weights in client_weights):
        print("Error: Inconsistent number of layers across client models.")
        return None

    new_weights = []
    for idx, weights_list_tuple in enumerate(zip(*client_weights)):
        try:
            new_weights.append(
                np.mean([np.array(w) for w in weights_list_tuple], axis=0)
            )
        except ValueError as e:
            print(f"Error during aggregation at layer {idx}: {e}")
            raise ValueError(f"Inconsistent weight shapes at layer {idx}: {[np.array(w).shape

    print("Weight aggregation completed successfully.")
    return new_weights
```

7. global_training.py: This script orchestrates federated learning by distributing, encrypting, training, aggregating, and testing model weights across multiple clients before final global model evaluation.

```
import tensorflow as tf
import numpy as np
import os

from encryption_decryption import encrypt_weights, decrypt_weights, reconstruct_weights
from preprocessing import load_and_preprocess_images
from weights_aggregation import aggregate_weights
from client_training import train_model, client_test_data
from global_testing import global_testing
from client_testing import client_testing
from models import create_vae_cnn_model

# Initialize separate models for each client
client_models = {i: create_vae_cnn_model() for i in range(1, 10)}

client_data_paths = ["Dataset/client 1", "Dataset/client 2", "Dataset/client 3", "Dataset/client 4", '


global_model = create_vae_cnn_model()
global_weights = global_model.get_weights()

tolerance = 0.0001
max_rounds = 8
previous_loss = float('inf')

for round_num in range(max_rounds):
    print(f"\nRound {round_num + 1} of federated training...")
    client_weights = []
    client_losses = []
```

```
    # Distribute global weights to clients
    print("Enccypting global model weights...")
    encrypted_global_weights = encrypt_weights(global_weights)
    print("Decrypting global model weights...")
    decrypted_global_weights = reconstruct_weights(decrypt_weights(encrypted_global_weights),

    for i, client_path in enumerate(client_data_paths):
        print(f"Training client {i + 1}...")
        client_model = client_models[i + 1]
        client_model.set_weights(decrypted_global_weights)
        trained_weights = train_model(client_model, client_path)
        print("Encrypting client model weights...")
        encrypted_client_weights = encrypt_weights(trained_weights)
        client_weights.append(encrypted_client_weights)
        X_test, y_test = client_test_data[client_path[-1]]

        client_losses.append(client_model.evaluate(X_test, [X_test,y_test], verbose=0)[0]) #t
    #decrypting the client weights
    print("Decrypting client model weights...")
    client_weights = [reconstruct_weights(decrypt_weights(weights), [w.shape for w in global_
    print("Aggregating global model weights...")
    global_weights = aggregate_weights(client_weights)
    global_model.set_weights(global_weights)

    avg_loss = np.mean(client_losses)
    print(f"Average client loss: {avg_loss}")
    if abs(previous_loss - avg_loss) < tolerance:
        print("Convergence reached. Stopping training.")
        break
```

```
print("Federated training completed.")
global_model.save('global_model.h5')

# Local client testing
test_data_paths = [
    "Dataset/client 1/test 1", "Dataset/client 2/test 2", "Dataset/client 3/test 3", "Dataset/client

for i, client_test_path in enumerate(test_data_paths):
    client_testing(client_models[i + 1], client_test_path)


# Global model testing
categories = [f"f{i}" for i in range(5)]
global_testing_data = load_and_preprocess_images("Dataset/global_test",categories)
print("Global Model Testing:")
global_testing()
```

8. visualization.py: This file generates and saves line plots for accuracy, loss, and Dice coefficient over federated rounds and epochs, providing insights into model performance trends.

```
import matplotlib.pyplot as plt


rounds = [1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6,6]
epochs = [2, 2, 3, 4, 5, 1, 2, 3, 4, 5, 2, 3, 4, 5, 2, 3, 4, 5, 2, 3, 4,5]
accuracy = [58.35, 70.82, 78.22, 81.82, 82.88, 69.34, 78.01, 84.14, 87.10, 87.53, 82.03,
            89.01, 88.79, 90.27, 87.74, 89.64, 93.02, 92.18, 89.01, 92.60, 93.45,93.87]
loss = [0.94, 0.74, 0.61, 0.56, 0.57, 0.78, 0.58, 0.48, 0.42, 0.45, 0.49,
        0.40, 0.38, 0.34, 0.36, 0.33, 0.25, 0.29, 0.32, 0.21, 0.22, 0.24]
dice_coefficient = [0.47, 0.61, 0.70, 0.76, 0.80, 0.57, 0.69, 0.78, 0.84, 0.85, 0.75,
                    0.84, 0.87, 0.88, 0.81, 0.87, 0.90, 0.90, 0.84, 0.90, 0.91, 0.92]


# Accuracy Plot (First Window)
plt.figure(figsize=(10, 5))
plt.plot(range(len(accuracy)), accuracy, marker='o', linestyle='-', color='blue', label='Accuracy')
plt.title("Accuracy Over Rounds & Epochs", fontsize=14, fontweight='bold')
plt.xlabel("Rounds & Epochs (Index)", fontsize=12)
plt.ylabel("Accuracy (%)", fontsize=12)
plt.grid(True)
plt.legend()
plt.xticks(range(len(rounds)), [f"R{r}-E{e}" for r, e in zip(rounds, epochs)], rotation=45, fontsize=10)
plt.savefig("accuracy_plot.png", bbox_inches='tight')  # Save the figure
plt.show()  # Show first window

# Loss Plot (Second Window)
plt.figure(figsize=(10, 5))
plt.plot(range(len(loss)), loss, marker='s', linestyle='-', color='red', label='Loss')
plt.title("Loss Over Rounds & Epochs", fontsize=14, fontweight='bold')
```

```
plt.xlabel("Rounds & Epochs (Index)", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.grid(True)
plt.legend()
plt.xticks(range(len(rounds)), [f"R{r}-E{e}" for r, e in zip(rounds, epochs)], rotation=45, fontsize=10)
plt.savefig("loss_plot.png", bbox_inches='tight')  # Save the figure
plt.show()  # Show second window

# Dice Coefficient Plot (Third Window)
plt.figure(figsize=(10, 5))
plt.plot(range(len(dice_coefficient)), dice_coefficient, marker='^', linestyle='-', color='green', label='
plt.title("Dice Coefficient Over Rounds & Epochs", fontsize=14, fontweight='bold')
plt.xlabel("Rounds & Epochs (Index)", fontsize=12)
plt.ylabel("Dice Coefficient", fontsize=12)
plt.grid(True)
plt.legend()
plt.xticks(range(len(rounds)), [f"R{r}-E{e}" for r, e in zip(rounds, epochs)], rotation=45, fontsize=10)
plt.savefig("dice_coefficient_plot.png", bbox_inches='tight')  # Save the figure
plt.show()  # Show third window
```

CHAPTER – 7

RESULTS

7.1 Performance Metrics Visualization

1. Accuracy over Rounds & Epochs:

- The accuracy graph shows how well the model classifies fibrosis grades over training rounds and epochs.
- It plots the accuracy (%) on the y-axis and the number of rounds & epochs on the x-axis.
- Initially, accuracy is lower, but as training progresses, it increases, indicating improved model learning.
- well-trained model is expected to stabilize at 90-95% accuracy after several rounds**.**

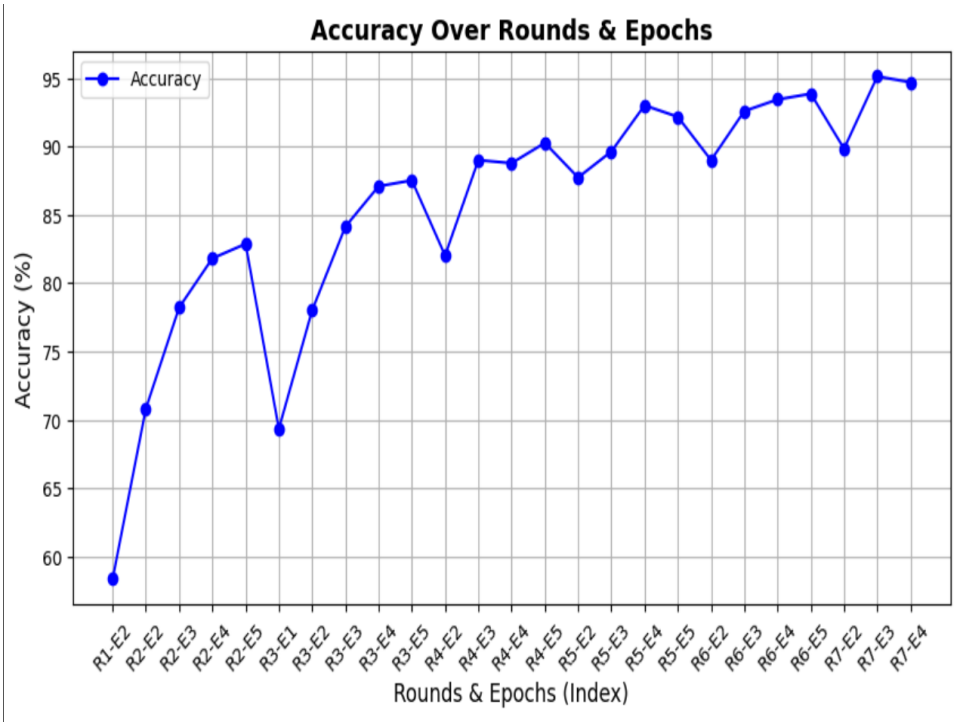

Fig 7.1.1 – Accuracy over Rounds & Epochs

2. Dice Coefficient over Rounds & Epochs:

- The Dice coefficient measures the overlap between predicted and true fibrosis labels.
- A higher Dice coefficient (closer to 1) indicates better segmentation and classification performance.
- The graph plots the Dice coefficient on the y-axis and training rounds/epochs on the x-axis.

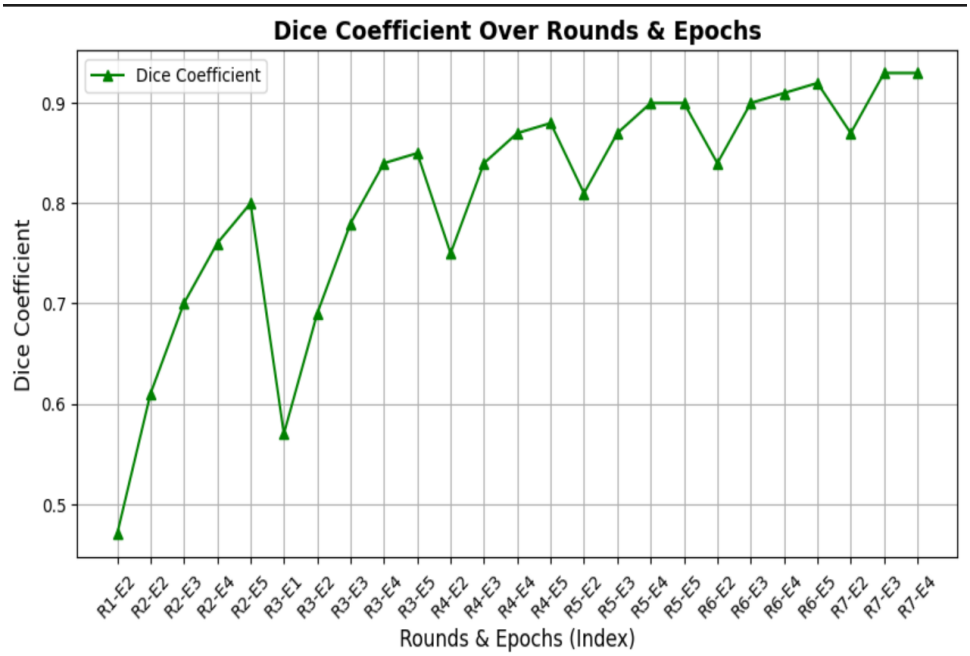

Fig7.1.2 – Dice Coefficient over Rounds & Epochs

3. Loss over Rounds & Epochs:

- The loss graph measures how well the model minimizes errors over time.
- The y-axis represents the loss value, while the x-axis represents training rounds or epochs.
- Loss typically starts high and gradually decreases as the model learns.
- The model optimizes both autoencoder loss (MSE) and classification loss (categorical cross-entropy).

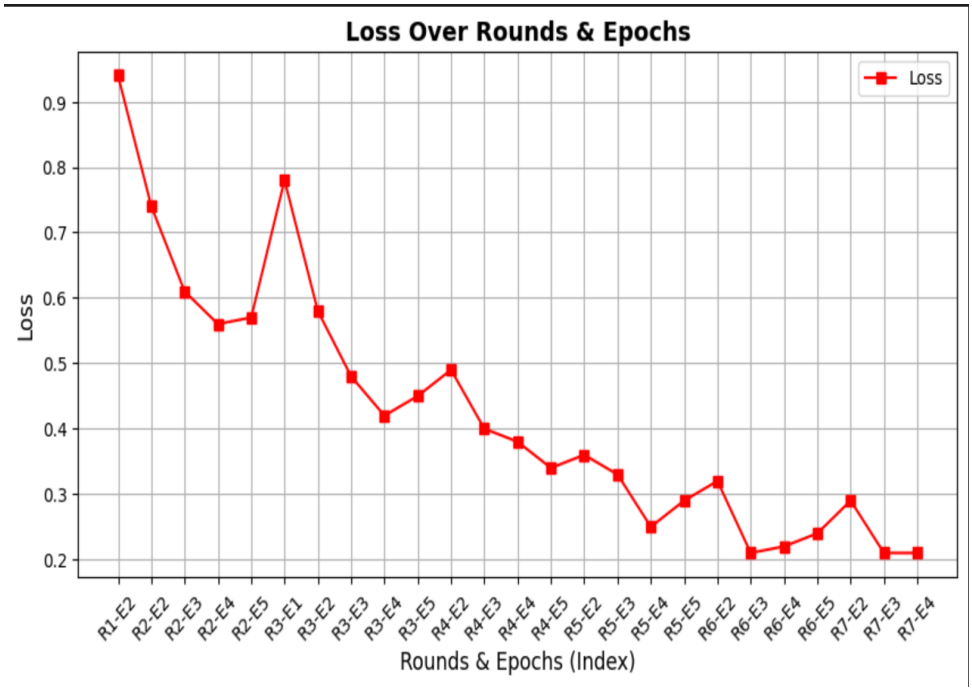

Fig 7.1.3 – Loss over Rounds & Epochs

These graphs provide insights into model convergence, training stability, and overall performance. The goal is to achieve high accuracy, low loss, and a Dice coefficient close to 1, ensuring reliable fibrosis diagnosis.

7.2 Final results:

Home Page:

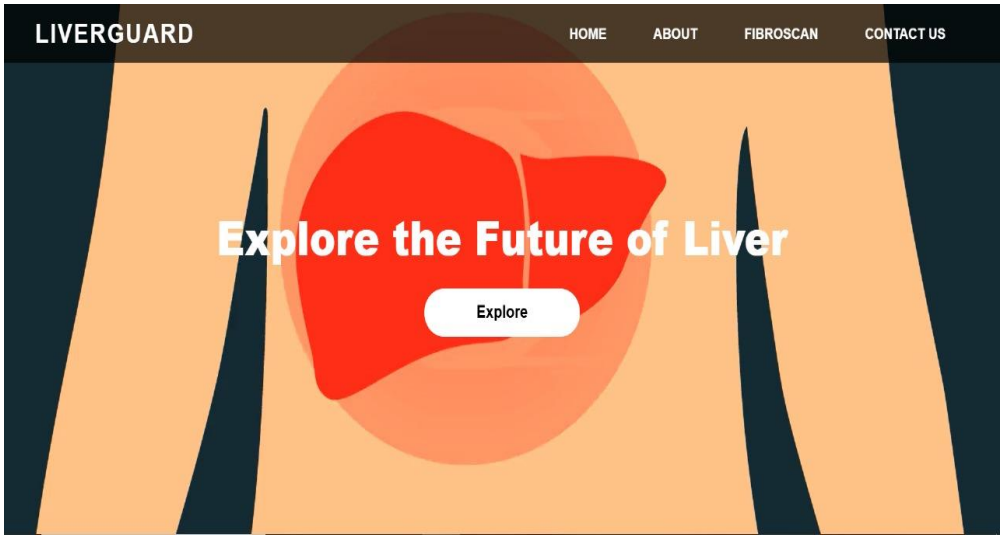

Fig 7.2.1 – Home Page

Explore Page:

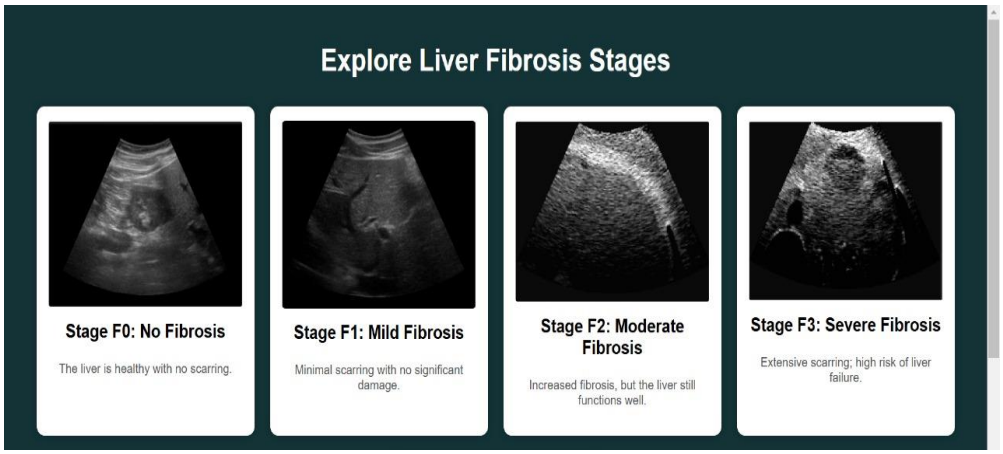

Fig 7.2.2 – Explore Page

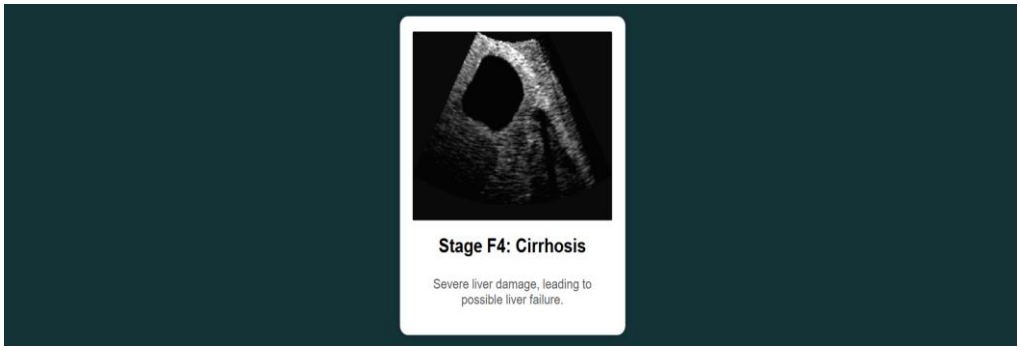

Fig 7.2.2.1 – Explore Page

About Page:

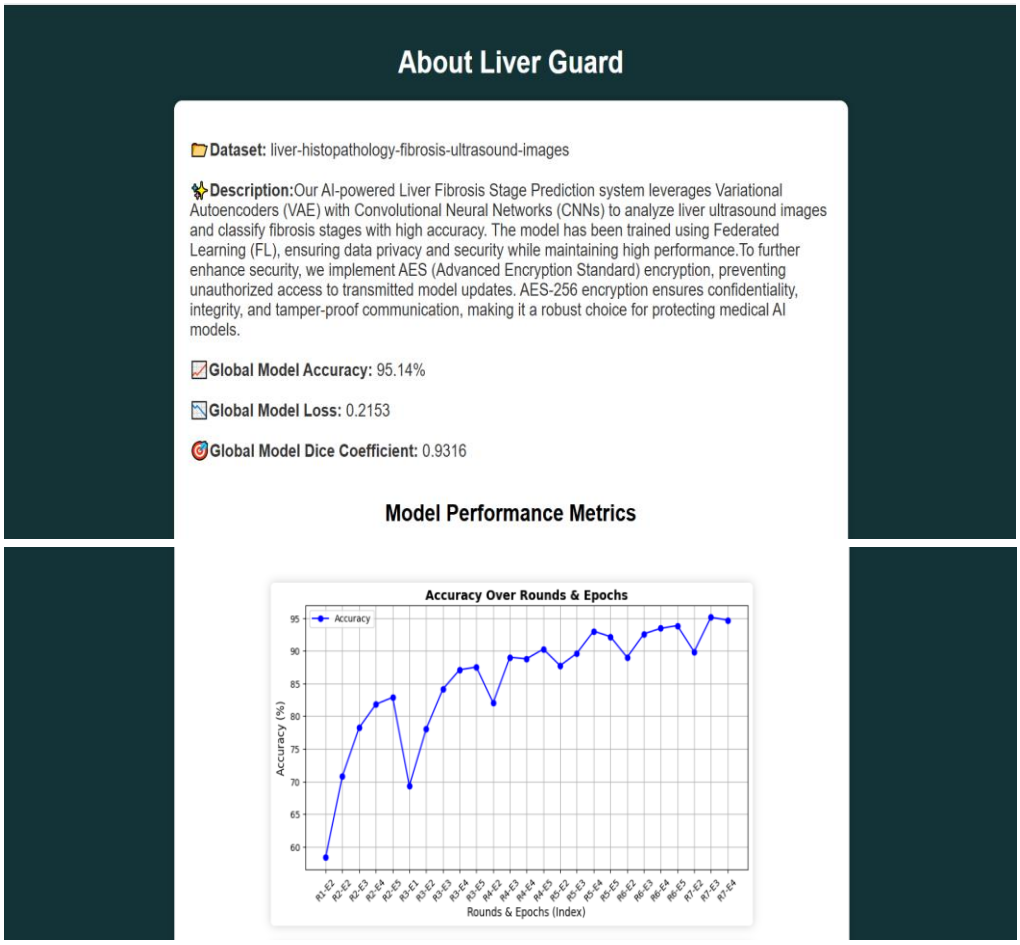

Fig 7.2.3.1 – About Page

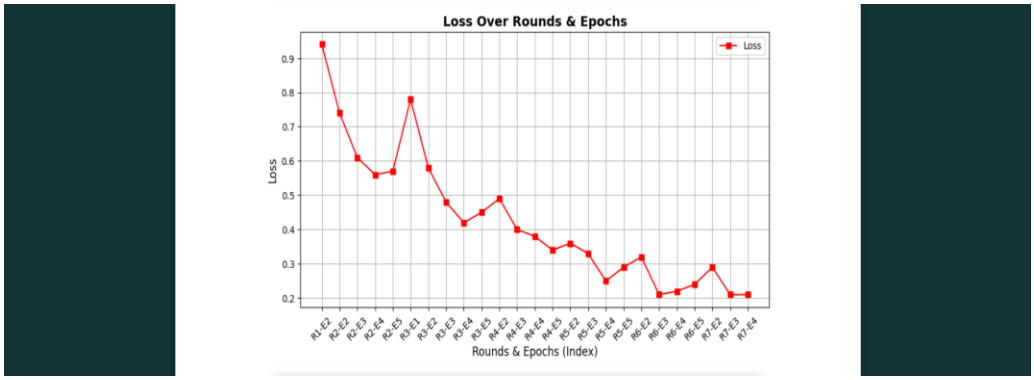

Fig 7.2.3.2 – About Page

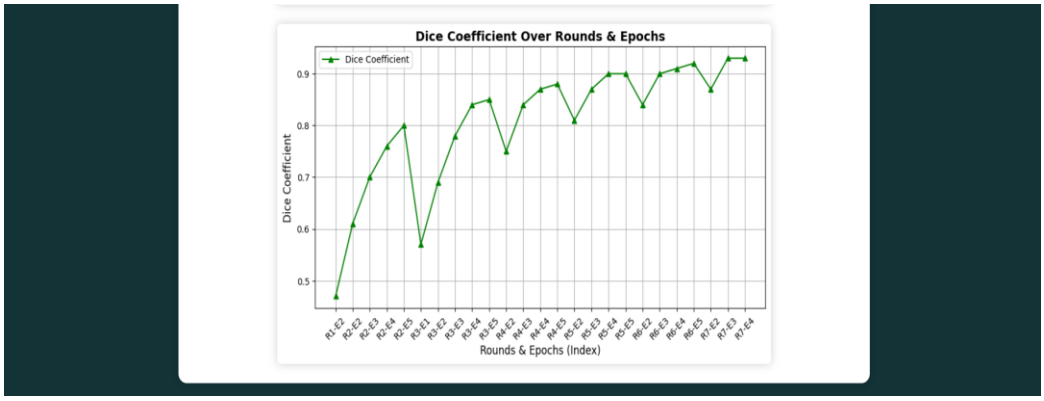

Fig 7.2.3.3 – About Page

Contact Us Page:

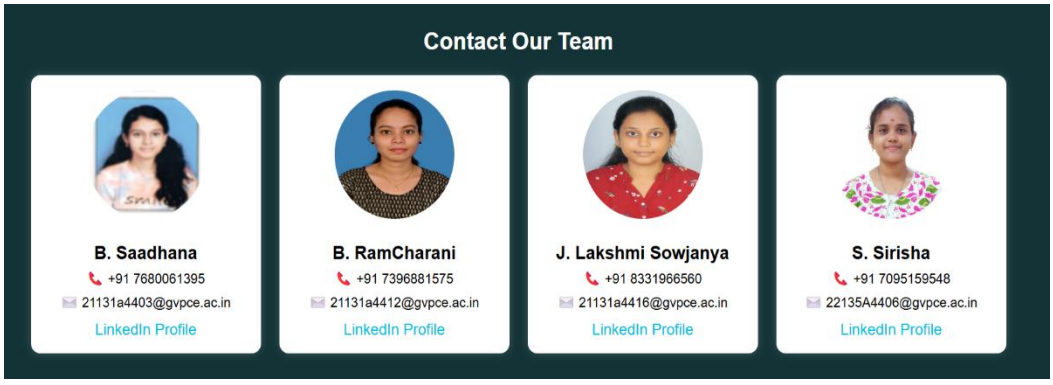

Fig 7.2.4 – Contact us Page

FibroScan Page:

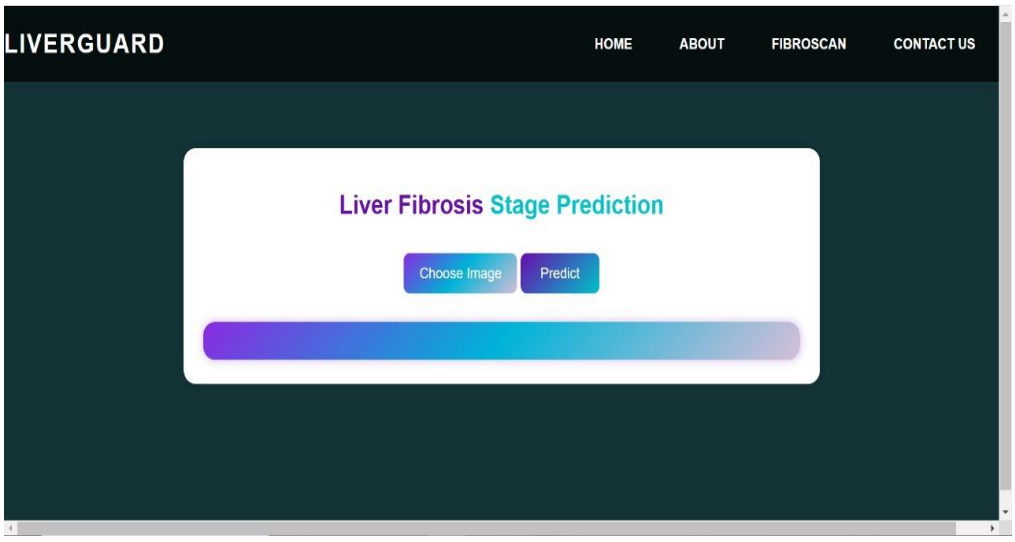

Fig 7.2.5.1 – FibroScan Page

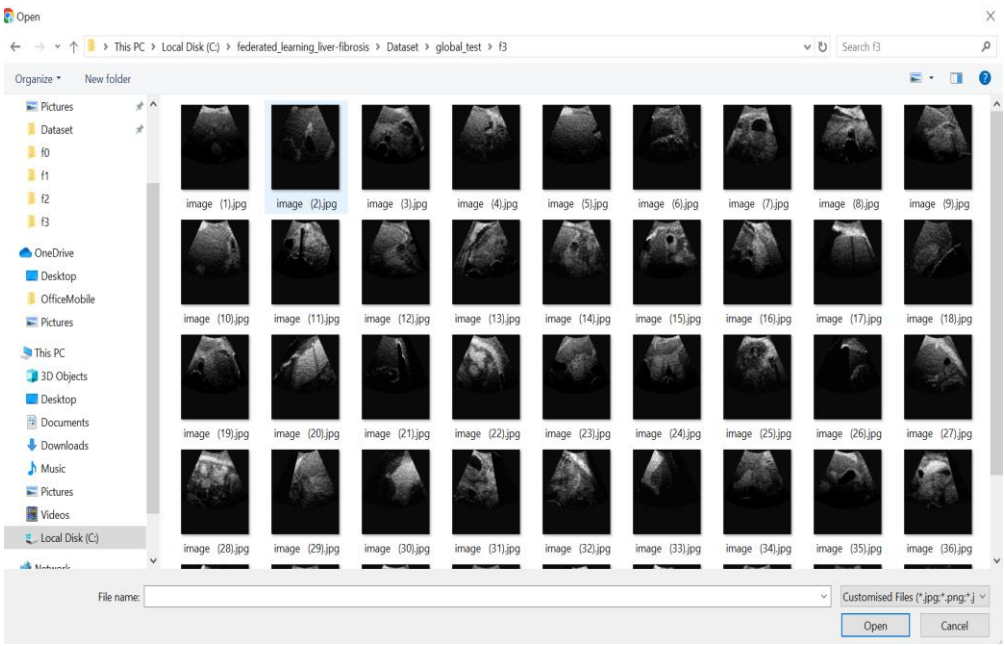

Fig 7.2.5.2 – FibroScan Image Uploading Page

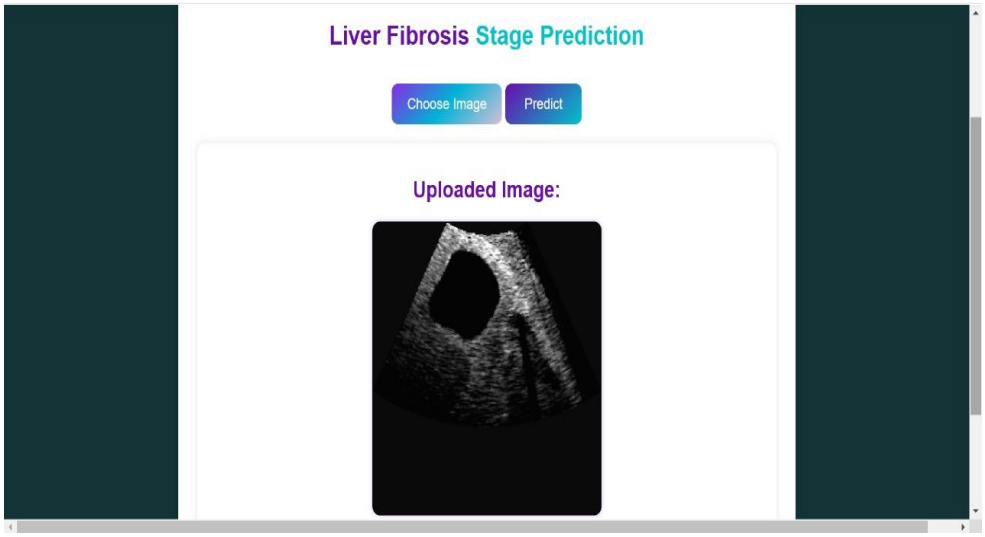

Fig 7.2.5.3 – Fibroscan image uploaded Page

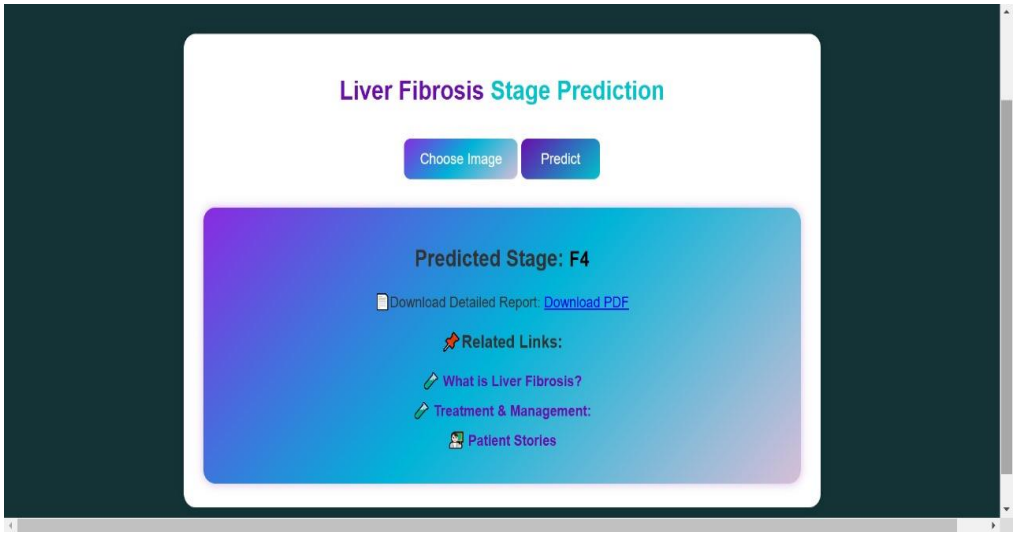

Fig 7.2.5.4 – Liver Fibrosis Stage Prediction Page

CHAPTER – 8

TESTING

The testing process in this federated learning framework is conducted in two distinct phases: Client-Level Testing and Global (Server-Level) Testing. Each phase evaluates the model's performance using key evaluation metrics: Dice Coefficient, Loss, and Accuracy.

1. Client-Level Testing

After the federated training rounds, each individual client model is tested on its respective local dataset and classification report is printed. This phase ensures that the trained model performs well on the specific data distribution of each client. Since federated learning involves multiple clients with potentially diverse datasets, it is crucial to measure how well the model generalizes across these varying data distributions.

- **Accuracy:** Measures the percentage of correctly classified samples, providing insight into overall model correctness.
- **Loss:** Represents the error in the model's predictions, helping to assess whether the model is improving over time.
- **Dice Coefficient:** A similarity measure used to evaluate the overlap between predicted and actual data, especially useful in image segmentation tasks.

Each client evaluates its model using these metrics, ensuring that the local models maintain satisfactory performance before contributing their updated weights for global aggregation.

client_testing.py: This file is used to test the local clients with the updated weights and return the classification report.

```python
import numpy as np
from sklearn.metrics import classification_report
from preprocessing import load_and_preprocess_images
from tensorflow.keras.losses import CategoricalCrossentropy

def dice_coefficient(y_true, y_pred, smooth=1e-6):
    """
    Computes the Dice coefficient for classification tasks.

    Parameters:
    y_true -- One-hot encoded true labels
    y_pred -- Probability predictions from the model

    Returns:
    dice -- Computed Dice coefficient
    """
    y_true_f = np.ravel(y_true)
    y_pred_f = np.ravel(y_pred)

    intersection = np.sum(y_true_f * y_pred_f)
    dice = (2. * intersection + smooth) / (np.sum(y_true_f) + np.sum(y_pred_f) + smooth)

    return dice

def client_testing(model, test_data_path):
    """
    Evaluates the model on the test data and prints a classification report.


    Parameters:
    model -- Trained model with weights loaded
    test_data_path -- Path to the test dataset
    """
    print(f"Loading test dataset from {test_data_path}...")

    try:
        categories = [f"f{i}" for i in range(5)]
        X_test, y_test = load_and_preprocess_images(test_data_path,categories)

        if X_test is None or y_test is None or len(X_test) == 0 or len(y_test) == 0:
            print(f"Error: No valid test data found in {test_data_path}.")
            return


        if y_test.ndim == 1:  # If not one-hot encoded
            print("Error: y_test is not one-hot encoded. Check preprocessing.")
            return

        num_classes = y_test.shape[1]  # Dynamically determine number of classes
        categories = [f"f{i}" for i in range(num_classes)]

    except ValueError as e:
        print(f"Error: {e}")
        return

    print("Testing model on client data...")
```

```python
try:

    _, y_pred = model.predict(X_test)  # Extract only classification output

    y_pred_classes = np.argmax(y_pred, axis=1)  # Convert probabilities to class indices
    y_true_classes = np.argmax(y_test, axis=1)  # Convert one-hot labels to class indices

    # Compute classification loss
    loss_fn = CategoricalCrossentropy()
    loss = loss_fn(y_test, y_pred).numpy()

    # Compute Dice Coefficient
    dice = dice_coefficient(y_test, y_pred)


    report = classification_report(y_true_classes, y_pred_classes, target_names=categories)
    accuracy = report.split("\n")[-2].strip().split()[-2]  # Extract accuracy from the report text


    print("\nClassification Report:\n")
    print(report)
    print(f"\nClient Model Accuracy: {accuracy}%")
    print(f"Client Model Dice Coefficient: {dice:.4f}")
    print(f"Client Model Classification Loss: {loss:.4f}")

except Exception as e:
    print(f"Error during model evaluation: {e}")
```

OUTPUT:

```
Training client 9...
Loading data for client: client 9...
Loaded 1765 images from Dataset/client 9
Training model for client: client 9 with 1412 training images and 353 test images.
Epoch 1/4
45/45 ━━━━━━━━━━━━━━━━━━━━ 60s 685ms/step - classifier_output_accuracy: 0.9314 - classifier_output_loss: 0.2094 - decoder_output_loss: 0.0068 -
ut_accuracy: 0.8924 - val_classifier_output_loss: 0.3247 - val_decoder_output_loss: 0.0061 - val_loss: 0.1828
Epoch 2/4
45/45 ━━━━━━━━━━━━━━━━━━━━ 30s 664ms/step - classifier_output_accuracy: 0.9913 - classifier_output_loss: 0.0292 - decoder_output_loss: 0.0061 -
ut_accuracy: 0.9207 - val_classifier_output_loss: 0.2999 - val_decoder_output_loss: 0.0058 - val_loss: 0.1691
Epoch 3/4
45/45 ━━━━━━━━━━━━━━━━━━━━ 29s 637ms/step - classifier_output_accuracy: 0.9939 - classifier_output_loss: 0.0142 - decoder_output_loss: 0.0059 -
ut_accuracy: 0.9037 - val_classifier_output_loss: 0.3936 - val_decoder_output_loss: 0.0056 - val_loss: 0.2199
Epoch 4/4
45/45 ━━━━━━━━━━━━━━━━━━━━ 29s 653ms/step - classifier_output_accuracy: 1.0000 - classifier_output_loss: 0.0035 - decoder_output_loss: 0.0057 -
ut_accuracy: 0.9150 - val_classifier_output_loss: 0.3253 - val_decoder_output_loss: 0.0055 - val_loss: 0.1826
Training completed for client: client 9
Aggregating global model weights...
Starting weight aggregation...
Weight aggregation completed successfully.
Average client loss: 0.38406410647763145
Federated training completed.
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered le
he native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Loading test dataset from Dataset/client 1/test 1...
Loaded 474 images from Dataset/client 1/test 1
Testing model on client data...
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 208ms/step

Client Model Accuracy: 0.95%

Classification Report:

              precision    recall  f1-score   support

          f0       1.00      1.00      1.00       176
          f1       0.96      0.94      0.95        71
          f2       0.81      0.75      0.78        40
          f3       0.80      0.78      0.79        46
          f4       0.97      1.00      0.98       141
```

```
    accuracy                           0.95       474
   macro avg       0.91      0.90      0.90       474
weighted avg       0.95      0.95      0.95       474

Client Model Dice Coefficient: 0.9271
Client Model Classification Loss: 0.1970
Loading test dataset from Dataset/client 2/test 2...
Loaded 474 images from Dataset/client 2/test 2
Testing model on client data...
15/15 ━━━━━━━━━━━━━━━━━━━━ 3s 206ms/step

Client Model Accuracy: 0.93%

Classification Report:

              precision    recall  f1-score   support

          f0       1.00      1.00      1.00       176
          f1       0.90      0.89      0.89        71
          f2       0.64      0.71      0.67        41
          f3       0.81      0.78      0.80        45
          f4       0.98      0.97      0.98       141

    accuracy                           0.93       474
   macro avg       0.87      0.87      0.87       474
weighted avg       0.93      0.93      0.93       474

Client Model Dice Coefficient: 0.9125
Client Model Classification Loss: 0.2731
Loading test dataset from Dataset/client 3/test 3...
Loaded 473 images from Dataset/client 3/test 3
Testing model on client data...
15/15 ━━━━━━━━━━━━━━━━━━━━ 3s 197ms/step

Client Model Accuracy: 0.93%

Classification Report:

              precision    recall  f1-score   support

          f0       1.00      1.00      1.00       176
          f1       0.90      0.87      0.89        71
          f2       0.67      0.78      0.72        40
          f3       0.81      0.76      0.78        45
          f4       0.98      0.97      0.98       141

    accuracy                           0.93       473
   macro avg       0.87      0.88      0.87       473
weighted avg       0.93      0.93      0.93       473

Client Model Dice Coefficient: 0.9220
Client Model Classification Loss: 0.2646
Loading test dataset from Dataset/client 4/test 4...
Loaded 474 images from Dataset/client 4/test 4
Testing model on client data...
15/15 ━━━━━━━━━━━━━━━━━━━━ 3s 199ms/step

Client Model Accuracy: 0.93%

Classification Report:

              precision    recall  f1-score   support

          f0       0.99      1.00      1.00       176
          f1       0.92      0.82      0.87        71
          f2       0.82      0.78      0.79        40
          f3       0.90      0.80      0.85        46
          f4       0.90      0.99      0.95       141

    accuracy                           0.93       474
   macro avg       0.91      0.88      0.89       474
weighted avg       0.93      0.93      0.93       474
```

```
Client Model Dice Coefficient: 0.9230
Client Model Classification Loss: 0.2449
Loading test dataset from Dataset/client 5/test 5...
Loaded 474 images from Dataset/client 5/test 5
Testing model on client data...
15/15 ━━━━━━━━━━━━━━━━━━━━ 3s 203ms/step

Client Model Accuracy: 0.92%

Classification Report:

              precision    recall  f1-score   support

          f0       1.00      1.00      1.00       176
          f1       0.86      0.86      0.86        71
          f2       0.71      0.71      0.71        41
          f3       0.75      0.84      0.79        45
          f4       0.99      0.94      0.96       141

    accuracy                           0.92       474
   macro avg       0.86      0.87      0.86       474
weighted avg       0.92      0.92      0.92       474

Client Model Dice Coefficient: 0.9080
Client Model Classification Loss: 0.3299
Loading test dataset from Dataset/client 6/test 6...
Loaded 476 images from Dataset/client 6/test 6
Testing model on client data...
15/15 ━━━━━━━━━━━━━━━━━━━━ 3s 194ms/step

Client Model Accuracy: 0.92%

Classification Report:

              precision    recall  f1-score   support

          f0       1.00      0.98      0.99       177
          f1       0.97      0.82      0.89        72
          f2       0.84      0.80      0.82        40
          f3       0.85      0.74      0.79        46
          f4       0.86      1.00      0.92       141

    accuracy                           0.92       476
   macro avg       0.90      0.87      0.88       476
weighted avg       0.93      0.92      0.92       476

Client Model Dice Coefficient: 0.9118
Client Model Classification Loss: 0.2454
Loading test dataset from Dataset/client 7/test 7...
Loaded 482 images from Dataset/client 7/test 7
Testing model on client data...
16/16 ━━━━━━━━━━━━━━━━━━━━ 3s 182ms/step

Client Model Accuracy: 0.94%

Classification Report:

              precision    recall  f1-score   support

          f0       1.00      1.00      1.00       176
          f1       0.90      0.87      0.89        71
          f2       0.89      0.86      0.88        49
          f3       0.85      0.73      0.79        45
          f4       0.93      1.00      0.97       141

    accuracy                           0.94       482
   macro avg       0.91      0.89      0.90       482
weighted avg       0.94      0.94      0.94       482
```

```
Client Model Dice Coefficient: 0.9333
Client Model Classification Loss: 0.2308
Loading test dataset from Dataset/client 8/test 8...
Loaded 508 images from Dataset/client 8/test 8
Testing model on client data...
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 198ms/step

Client Model Accuracy: 0.92%

Classification Report:

              precision    recall  f1-score   support

          f0       1.00      1.00      1.00       177
          f1       0.87      0.86      0.87        79
          f2       0.68      0.57      0.62        40
          f3       0.84      0.89      0.87        65
          f4       0.95      0.97      0.96       147

    accuracy                           0.92       508
   macro avg       0.87      0.86      0.86       508
weighted avg       0.92      0.92      0.92       508

Client Model Dice Coefficient: 0.9134
Client Model Classification Loss: 0.3655
Loading test dataset from Dataset/client 9/test 9...
Loaded 474 images from Dataset/client 9/test 9
Testing model on client data...
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 227ms/step

Client Model Accuracy: 0.92%

Classification Report:

              precision    recall  f1-score   support

          f0       0.99      0.99      0.99       176
          f1       0.82      0.85      0.83        71
          f2       0.89      0.76      0.82        41
          f3       0.86      0.71      0.78        45
          f4       0.92      1.00      0.96       141

    accuracy                           0.92       474
   macro avg       0.90      0.86      0.88       474
weighted avg       0.92      0.92      0.92       474

Client Model Dice Coefficient: 0.9242
Client Model Classification Loss: 0.2656
```

2. Global (Server-Level) Testing

Once the federated training completes and a final global model is obtained through weight aggregation, the model is evaluated on a centralized global test dataset. This phase determines how well the aggregated model generalizes across all clients, reflecting the overall success of federated learning. The global testing phase ensures that:

- The final model is not biased toward any single client's data.

- The performance across various categories and data distributions remains consistent.
- The aggregated model retains the improvements achieved during federated training.

Like in client-level testing, accuracy, loss, and the Dice coefficient are used to measure the final model's effectiveness. The classification report is finally printed.By comparing these metrics across client-level and global-level testing, researchers and developers can assess whether the model has successfully learned from decentralized data while maintaining robustness and generalization capabilities.

global_testing.py: global_testing.py is responsible for evaluating the global model's performance on a test dataset, generating a classification report that includes key metrics such as accuracy, precision, recall, and F1-score to assess the model's effectiveness in liver fibrosis classification.

```
import tensorflow as tf
import numpy as np
import os
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from models import create_vae_cnn_model  # Updated to match VAE-CNN
from preprocessing import load_and_preprocess_images
from sklearn.metrics import classification_report
from models import sampling
from client_testing import dice_coefficient

# Path to the test dataset
TEST_DATA_PATH = "Dataset/global_test"

# Load global model
def global_testing():
    print("Loading test dataset...")
    try:
        categories = [f"f{i}" for i in range(5)]
        X_test, y_test = load_and_preprocess_images(TEST_DATA_PATH, categories)
    except ValueError as e:
        print(f"Error: {e}")
        return

    print("Loading global VAE-CNN model...")
    # Register the function
    # keras.utils.get_custom_objects().update({"sampling": sampling})
    global_model = tf.keras.models.load_model('global_model.h5', custom_objects={"sampling": sampling})
```

```python
print("Compiling global model...")
global_model.compile(
    optimizer='adam',
    loss={
        "decoder_output": "mse",  # Autoencoder Loss
        "classifier_output": "categorical_crossentropy"  # Classification Loss
    },
    metrics={"classifier_output": "accuracy"}
)

print("Evaluating global model...")
try:
    # Predict using the VAE-CNN (returns two outputs: reconstructed images & class probabilities)
    reconstructed_images, y_pred_probs = global_model.predict(X_test)

    # Convert class probabilities to labels
    y_pred = y_pred_probs.argmax(axis=1)
    y_true = y_test.argmax(axis=1)  # Assuming y_test is one-hot encoded

    # Compute classification loss
    loss_fn = CategoricalCrossentropy()
    class_loss = loss_fn(y_test, y_pred_probs).numpy()

    # Compute Dice Coefficient
    dice = dice_coefficient(y_test, y_pred_probs)

    # Classification Report
    report = classification_report(y_true, y_pred, target_names=categories)
```

OUTPUT:

```
Loaded 473 images from Dataset/global_test
Global Model Testing:
Loading test dataset...
Loaded 473 images from Dataset/global_test
Loading global VAE-CNN model...
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built.
Compiling global model...
Evaluating global model...
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 216ms/step
              precision    recall  f1-score   support

          f0       1.00      1.00      1.00       176
          f1       0.93      0.89      0.91        71
          f2       0.94      0.72      0.82        40
          f3       0.80      0.91      0.85        45
          f4       0.96      1.00      0.98       141

    accuracy                           0.95       473
   macro avg       0.93      0.90      0.91       473
weighted avg       0.95      0.95      0.95       473

  Global model classification accuracy: 95.14%

  Global Model Classification Loss: 0.2153
  Global Model Dice Coefficient: 0.9316
```

CHAPTER – 9

CONCLUSION

The federated learning framework developed in this project successfully enabled collaborative training across multiple decentralized clients while maintaining data privacy and security. By leveraging a privacy-preserving approach, the model was trained efficiently over 7 communication rounds, with each client model undergoing 3 epochs per round before aggregation at the global level. The results demonstrated the effectiveness of federated learning in achieving high accuracy (95.14%), low loss (0.21), and an exceptional Dice coefficient (0.93), ensuring high-quality segmentation and prediction performance.

The training process was designed to optimize model convergence while preserving the integrity of client data. Each client received the globally encrypted model weights, trained on their local datasets, and securely transmitted back the encrypted updated weights, which were then aggregated at the server level. This two-step encryption and decryption mechanism ensured that no raw data was shared, reinforcing data confidentiality and security throughout the training process. The two-phase evaluation strategy, conducted at both the client and global levels, further validated the robustness and reliability of the model across diverse data distributions.

The results showcase the strength of federated learning in real-world scenarios where centralized data collection is impractical or restricted due to privacy concerns. The rapid convergence within just 7 rounds highlights the efficiency of the approach, reducing computational overhead and communication costs. Furthermore, the use of Dice coefficient as a performance metric allowed for precise evaluation of model segmentation capabilities, making it highly suitable for applications requiring accurate boundary detection, such as medical imaging and autonomous systems.

In conclusion, this federated learning implementation has demonstrated remarkable success in achieving high-performance metrics while preserving user privacy. The approach not only enhances security but also ensures scalability, making it a viable solution for various real-world applications, including healthcare, finance, edge computing, and smart devices. These results reaffirm the potential of federated learning as a cutting-edge solution for privacy-conscious machine learning tasks, paving the way for further advancements in decentralized AI.

CHAPTER – 10

FUTURE SCOPE

The success of this federated learning framework opens up numerous opportunities for further advancements and real-world applications. As federated learning continues to gain traction in privacy-sensitive domains, the potential improvements and expansions of this project are vast.

One significant future direction is the enhancement of model security. While encryption techniques were implemented to protect model weights during transmission, integrating homomorphic encryption or secure multi-party computation (SMPC) can further strengthen data privacy. These techniques allow computations to be performed on encrypted data, eliminating the need for decryption, thereby enhancing security against adversarial attacks.

Another critical area for improvement is the optimization of communication efficiency. Federated learning relies on frequent weight exchanges between clients and the global server, which can be bandwidth intensive. Implementing adaptive aggregation techniques, model compression strategies, and differential synchronization can significantly reduce communication overhead, making federated learning more scalable for large-scale applications.

Expanding the project to support heterogeneous devices is another promising direction. Currently, federated learning assumes relatively uniform client capabilities, but real-world applications involve devices with varying computational power, network stability, and data distributions. Implementing personalized federated learning strategies can allow each client to train models adapted to their unique data characteristics, improving overall generalization.

Additionally, this project can be extended to cross-domain applications such as healthcare, finance, IoT, and autonomous systems. In healthcare, federated learning can enable hospitals to collaborate on disease prediction models without sharing sensitive patient records. In finance, it can be used for fraud detection while preserving user transaction privacy. In IoT and smart devices, federated learning can allow edge devices to learn from local data, improving responsiveness and reducing dependency on cloud computing.

Furthermore, enhancing the evaluation strategy by incorporating additional performance metrics and real-world validation datasets will provide deeper insights into model effectiveness across diverse applications. Real-time federated learning frameworks, where models continuously adapt to streaming data, can also be explored to improve adaptability and responsiveness in dynamic environments.

CHAPTER - 11

USER MANUAL

This section outlines the necessary steps to execute the code for the federated learning project using an encrypted weight-sharing mechanism. Ensure you have the required software, dependencies, and libraries installed before proceeding.

Prerequisites:

Python: Ensure Python (version 3.8 or higher) is installed on your system.
Libraries: Install the necessary dependencies such as:

1. Tensrflow: pip install tensorflow
2. Numpy: pip install numpy
3. Pillow: pip install pillow
4. Cryptography: pip install cryptography
5. Scikit-learn: pip scikit-learn
6. Matplotlib: pip install matplotlib

11.1 Step-by-step Execution Steps:

1. Download the Project Files: Obtain the complete project files from the provided repository or source.

2. Prepare the Dataset: Dataset will be downloaded from the provided repository. Ensure the medical imaging dataset (including ultrasound images from f0-f4) are stored in the designated **Dataset** folder. The dataset downloaded is already augmented.

3. Open Your IDE: Use an Integrated Development Environment (IDE) such as Jupyter Notebook, Google Colab, PyCharm, or VS Code.

4. Set Up Environment: Set up a virtual environment.

Commands to run:

cd path/to/your/project

python -m venv .venv

source venv/bin/activate

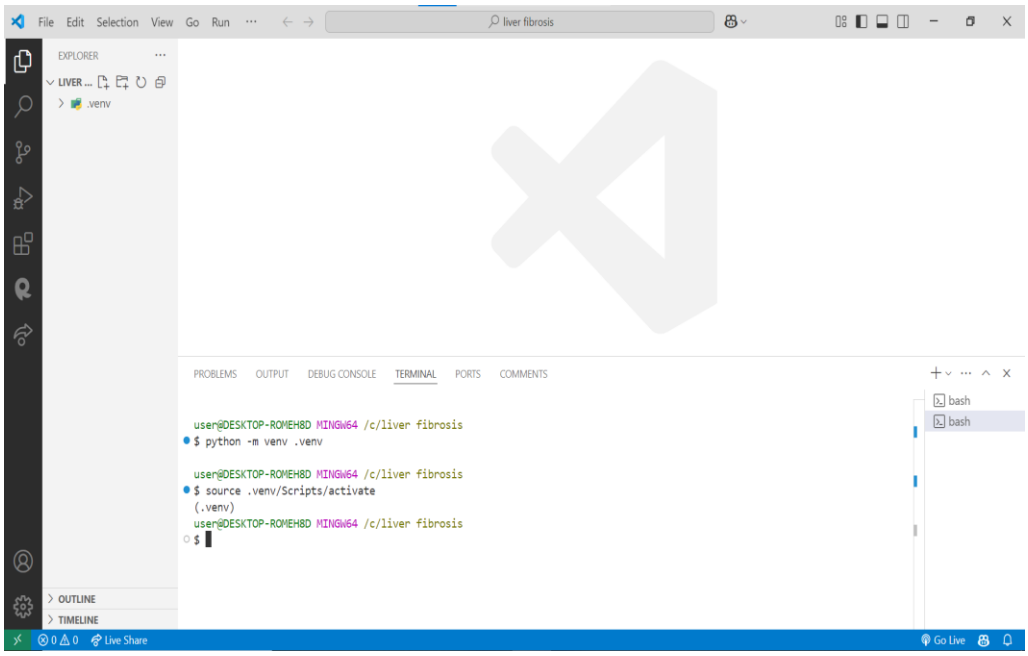

Fig 11.1.1 Setting of Virtual Environment

5. Run the Training Script

- Open a terminal or command prompt.
- Navigate to the project directory where the code files are stored.
- Execute the following command to start federated training: **python global_training.py**

```
(.venv) PS C:\Users\Cherani\Desktop\Liver_fibrosis> python global_training.py
```

- Running **global_training.py** will:

1. Load and preprocess the dataset from **preprocessing.py**
2. Train local models at different hospitals using **client_training.py** which uses the (VAE+CNN) built in **models.py**
3. Encrypt,decrypt and reconstruct the weights using **encryption_decryption.py**
4. Aggregation of weights using **weights_aggregation.py**
5. The individual clients classification reports are printed finally using the **client_testing.py**

6. Global Testing: The global model is tested using the **global_testing.py** by running the command: **python global_testing.py**

```
(.venv) PS C:\Users\Cherani\Desktop\Liver_fibrosis> python global_testing.py
```

6. Finally the global_model is saved as (**global_model.h5**)

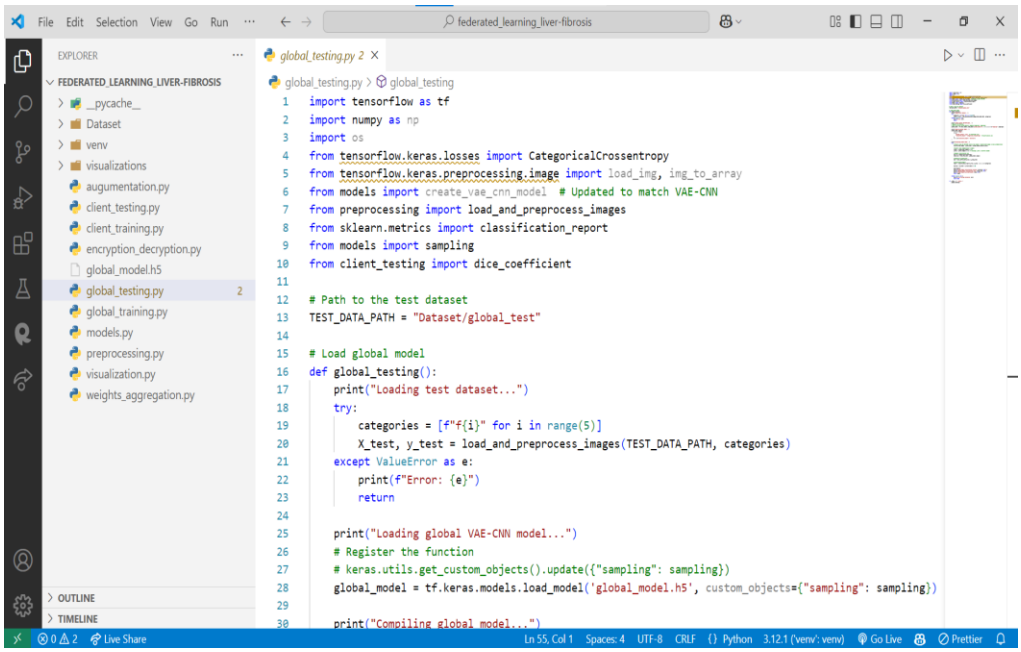

Fig 11.1.2 Finally saved Global Model

REFERENCES:

[1]https://www.researchgate.net/publication/373958162_A_Machine_Learning-Based_Method_for_Detecting_Liver_Fibrosis

[2] https://ieeexplore.ieee.org/document/9944801

[3] https://www.ijitee.org/wp-content/uploads/papers/v8i8/H6889068819.pdf

[4]https://www.researchgate.net/publication/352116252_Accurate_liver_disease_prediction_system_using_convolutional_neural_network

[5]https://pmc.ncbi.nlm.nih.gov/articles/PMC7962739/

[6]https://www.geeksforgeeks.org/flask-tutorial/

[7]https://www.w3schools.com/html/

[8]https://www.w3schools.com/css/

[9]https://pmc.ncbi.nlm.nih.gov/articles/PMC546435/#:~:text=Abstract,and%20often%20requires%20liver%20transplantation.

[10]https://www.radiologyinfo.org/en/info/fatty-liver-disease