SOFTWARE DESIGN SPECIFICATIONS DOCUMENT

FedFusion: Secure Collaboration using Enhanced Federated Learning

By

Muhammad Saadan 368560

Misbah Juwayriyyah 382416

Wasif Mehmood 367315

Supervisor:

Mohsin Kamal

Co-Supervisor:

Maajid Maqbool

Bachelor of Engineering in Software Engineering (2024-2025)

Department of Computing
School of Electrical Engineering and Computer Science
National University of Sciences & Technology

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

REVISION HISTORY

Name	Date	Reason for changes	Version
Encryption Scope	05/09/ 2024	Homomorphic encryption would be computationally expensive and time consuming to implement.	1.0.1
Image Support	05/09/ 2024	Text based support would be not fit as reference for evaluation and development of Federated Learning Platform	1.0.2
Dropping Data Recovery Attack	08/12/ 2024	Data Recovery Attack would be kept in future goals given complete working solution using Data Poisoning attack.	2.0.1
Automation of Dataset Information	08/12/ 2024	Frontend will be automated reducing the information to be entered by the users.	2.0.2

Table 1 Revision History

APPLICATION EVALUATION HISTORY

Comments (by committee)	Action Taken
Proposal Defense: Too many UN SDG mapping for the project. (By Madiha Khalid)	Narrowed down the SDG mapping from 3 goals to 1 goal only (9. Industry, Innovation and Infrastructure (Industry)).
Proposal Defense: Very specific UN SDG mapping for the project. (By Mehdi Hussain)	Finalized the SDG mapping from 1 to 2 goals (9. Industry, Innovation and Infrastructure (Industry) and 17. Partnerships for the Goals (Technology))
Proposal Defense: Homomorphic encryption scope reconsideration for the project. (By Madiha Khalid)	Adjusted the scope to keep Asymmetric Encryption as deliverable and Homomorphic Encryption as Future Goal depending on the progress.
Proposal Defense: High computation and parameters cost due to implementation based on LLMs. (By Mohsin Kamal)	Provided image-based support for federated learning along with initial text-based implementation.
Proposal Defense: Cite IEEE format-based references in the presentation as well. (By Maajid Maqbool)	Modified the presentation to adjust reference citations.
Mid Defense: Survey and Elicitations not provided in the SRS. Formatting of SRS was not consistent. (By Madiha Khalid)	Conducted Survey and Elicitations to include in the final version of SRS and fixed formatting consistency issues.
Mid Defense: Focus on attack implementation and simulation in platform. (By Mohsin Kamal)	Added federated learning-based attacks to the platform.
Mid Defense: Results were not shown clearly. (By Mehdi Hussain)	Due to required initial setup timing, saved results during development time separately to show in upcoming evaluation.

Table 2 Evaluation History

Supervised by

Mohsin Kamal

Signature_____

INTRODUCTION

The project FedFusion is focused on implementing a Federated Learning (FL) system. The scope of the project encompasses various aspects of federated learning, including data handling, model training, secure communication, and client-server interactions. Below is an overview of the project's scope, including its modules and functionalities:

PROJECT SCOPE

1. FEDERATED LEARNING FRAMEWORK: The project aims to simplify the implementation and experimentation with federated learning architectures, allowing researchers and practitioners to explore FL techniques effectively.
2. MULTI-CLIENT SUPPORT: The system supports multiple clients, simulating realistic federated learning scenarios where each client can train a model on its local data.
3. DATA HANDLING: The project includes functionalities to load datasets (e.g., CIFAR-10) from Hugging Face, preprocess them, and split them into training and testing sets for each client. It also allows for customization of datasets, enabling users to replace the default dataset with their own.
4. MODEL TRAINING: The project implements various model architectures (dense, image, text) using TensorFlow and Keras, allowing for flexibility in model design based on the type of data being used. It includes mechanisms for training models locally on client devices and aggregating updates on a central server.
5. SECURE COMMUNICATION: The project incorporates encryption techniques (RSA and AES) to secure the communication between clients and the server, ensuring that model parameters and updates are transmitted securely. It includes mechanisms for encrypting and decrypting model parameters to protect sensitive data during the federated learning process.
6. ANOMALY DETECTION AND METRICS TRACKING: The system implements methods for detecting anomalies in client updates using Z-score analysis and tracks training metrics (loss and accuracy) over multiple rounds of training.
7. CUSTOMIZATION AND EXTENSIBILITY: The framework is designed to be flexible and easily extendable, allowing users to modify components such as the model architecture, data loading, and training logic.

MODULES OVERVIEW

1. SERVER MODULE: The server module manages the central server for federated learning, coordinating the training process across multiple clients. It also implements a custom federated averaging strategy for aggregating client updates securely and efficiently.
2. CLIENT MODULE: The client module implements the client-side logic for federated learning, including data preparation, model training, and communication with the server. It also supports gradient leakage simulation and handles the encryption/decryption of model parameters.
3. DATA MODULE: The data module provides a set of utility functions designed to facilitate the loading, preparation, and processing of datasets for federated learning tasks. It integrates with Hugging Face's dataset library and includes functionalities for configuration management, data normalization, and serialization.

4. MODEL MODULE: The model module contains functions to build and compile various types of models (dense, image, text) using TensorFlow and Keras. These models are custom defined and can be extended in future for diverse support.
5. CRYPTO MODULE: The crypto module handles the encryption and decryption of model parameters and AES keys, ensuring secure communication between clients and the server.
6. ATTACK MODULE: The attack module simulates gradient leakage attacks and data poisoning, allowing for testing the robustness of the federated learning system against adversarial scenarios.
7. TESTING MODULE: The testing module provides functionality to test the encryption and decryption processes, ensuring that the security mechanisms are functioning correctly.

DESIGN METHODOLOGY AND SOFTWARE PROCESS MODEL

DESIGN METHODOLOGY:

The design methodology followed in the FedFusion project is primarily Object-Oriented Programming (OOP). This choice is justified for several reasons:

1. ENCAPSULATION: OOP allows for encapsulating data and behavior within classes. In our project the different components such as clients, servers, models, and data handling are represented as distinct classes. This encapsulation helps in managing complexity by grouping related functionalities and data together, making the codebase easier to understand and maintain.
2. REUSABILITY: OOP promotes code reusability through inheritance and polymorphism. For instance, the Client class can be extended to create specialized clients (e.g., GradientLeakageClient), allowing for the addition of new features without modifying existing code. This modularity facilitates the development of new functionalities, such as different attack simulations or model types, without duplicating code.
3. ABSTRACTION: OOP provides a way to abstract complex functionalities behind simple interfaces. For example, the 'create_flower_client' function abstracts the details of client initialization, allowing users to interact with a simple interface while hiding the underlying complexity. This abstraction simplifies the user experience and reduces the likelihood of errors.
4. MAINTAINABILITY: The use of OOP enhances maintainability by organizing code into manageable classes and modules. Changes to one part of the system (e.g., updating the model architecture) can be made with minimal impact on other parts, as long as the interfaces remain consistent. This is particularly important in our project FedFusion, where multiple components interact with each other.
5. REAL-WORLD MODELING: OOP aligns well with modeling real-world entities and relationships. In the context of federated learning, entities such as clients, servers, and datasets can be represented as objects, making the design intuitive and relatable. This helps developers and stakeholders understand the system's architecture more easily.

PROCESS MODEL (AGILE DEVELOPMENT):

The process model being followed in the FedFusion project can be characterized as Agile Development. This choice is justified for the following reasons:

1. ITERATIVE DEVELOPMENT: Agile emphasizes iterative development, allowing for incremental improvements and refinements. In the context of FedFusion, features can be developed, tested, and refined in small cycles, enabling quick feedback and adjustments based on user needs or testing results.
2. FLEXIBILITY AND ADAPTABILITY: Agile methodologies are inherently flexible, allowing teams to adapt to changing requirements. In a research-oriented project like FedFusion, where new techniques and algorithms may emerge, the ability to pivot and incorporate new ideas is crucial. Agile supports this adaptability by encouraging regular reassessment of priorities and goals.

3. COLLABORATION AND COMMUNICATION: Agile promotes collaboration among team members and stakeholders. Regular meetings, such as daily stand-ups or sprint reviews, facilitate communication and ensure that everyone is aligned on project goals. This collaborative environment is beneficial for a project as it involves multiple contributors with varying expertise.
4. USER -CENTRIC FOCUS: Agile methodologies prioritize user feedback and involvement throughout the development process. In the case of FedFusion, engaging with researchers and practitioners can help us identify pain points and desired features, leading to a more user-friendly and effective federated learning framework.
5. CONTINUOUS INTEGRATION AND TESTING: Agile encourages continuous integration and testing, which helps identify issues early in the development process. This is particularly important in a complex system like FedFusion, where multiple components must work together seamlessly. Regular testing ensures that new features do not introduce regressions or bugs.

SYSTEM OVERVIEW

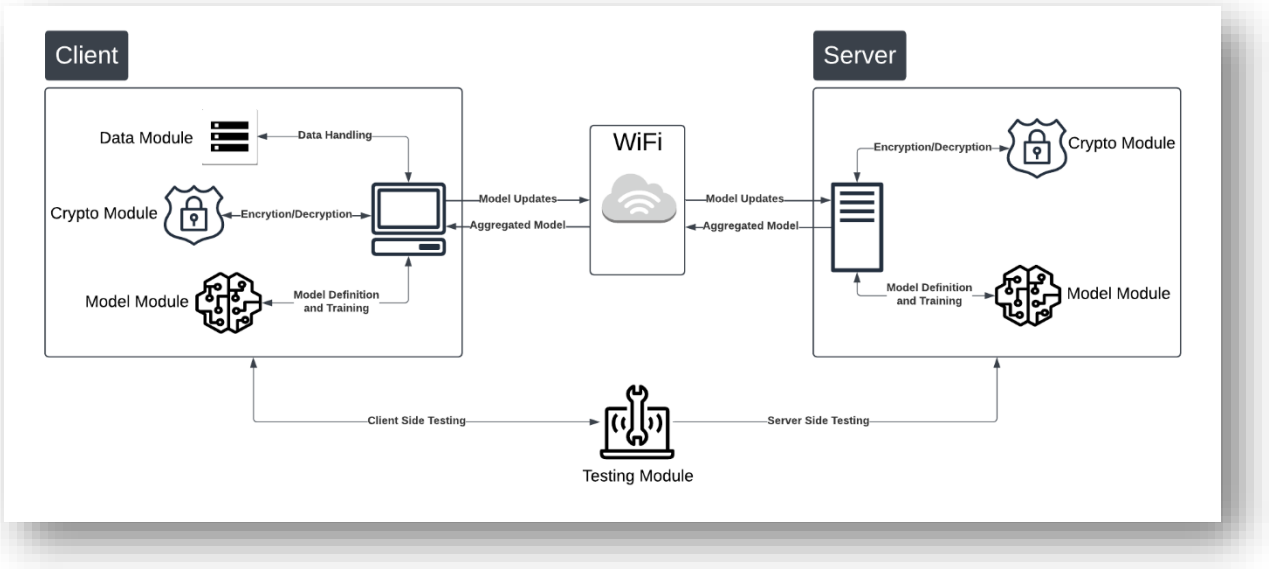
BACKGROUND INFORMATION:

Federated Learning is a distributed machine learning approach that enables multiple clients to collaboratively train a model while keeping their data decentralized. This approach is particularly relevant in scenarios where data privacy and security are paramount, such as in healthcare, finance, and personal devices. Traditional centralized machine learning methods require aggregating data in a single location, which can lead to privacy concerns and regulatory challenges.

1. GENERAL DESCRIPTION: FedFusion is an open-source framework designed to facilitate the implementation and experimentation of Federated Learning (FL) architectures. The primary functionalities of the project include:
2. MULTI-CLIENT FEDERATED LEARNING: The system supports multiple clients that can train machine learning models on their local datasets while collaborating with a central server to improve a global model. This simulates real-world scenarios where data privacy is crucial, as clients do not share their raw data.
3. MODEL TRAINING AND AGGREGATION: Clients can train models locally using their data and send model updates (gradients) to the server. The server aggregates these updates using techniques like Federated Averaging (FedAvg) to update the global model.
4. SECURE COMMUNICATION: The framework incorporates encryption mechanisms (RSA and AES) to ensure secure communication between clients and the server. This protects sensitive model parameters and updates from potential eavesdropping or tampering.
5. DATA HANDLING: FedFusion includes functionalities to load, preprocess, and split datasets (e.g., CIFAR-10) for training and testing. Users can customize datasets and model architectures to suit their specific needs.
6. ANOMALY DETECTION AND METRICS TRACKING: The system implements methods for detecting anomalies in client updates and tracks training metrics (loss and accuracy) over multiple rounds, providing insights into the training process.
7. CUSTOMIZATION AND EXTENSIBILITY: The framework is designed to be flexible and easily extendable, allowing users to modify components such as model architectures, data loading, and training logic.

ARCHITECTURAL DESIGN

ARCHITECTURE DIAGRAM:



‘

Figure 1 Architecture Diagram

DETAILED MAPPING TO ARCHITECTURE:

The architecture style chosen for FedFusion is a Client-Server Architecture. This architecture is suitable for federated learning systems where clients (participants) communicate with a central server to collaboratively train a model.

1. CLIENT MODULE: Part of Client: Represents the client-side of the architecture, where local data processing and model training occur.
2. SERVER MODULE: Part of Server: Represents the server-side of the architecture, responsible for aggregating updates and managing the global model.
3. MODEL MODULE: Part of Both: Provides model definitions and training logic that can be utilized by both clients and the server.
4. DATA MODULE: Part of Client: Supplies data to clients for training, ensuring that the data is preprocessed and ready for model input.
5. CRYPTO MODULE: Part of Both: Ensures secure communication between clients and the server by encrypting and decrypting model parameters.

6. TESTING MODULE: Part of Both: Validates the security mechanisms in place, ensuring that the encryption and decryption processes are functioning.
7. TEMPORARY MODULES: Note that some modules are temporarily created such as attack module which are used for carrying out research related to challenges faced by federated learning. Attack module serves as the utility module for creating a client-side attack and carrying out server-side defense. The reason for not considering the attack module is that its functionalities would later on be divided and included in the client module and server module accordingly.

DESIGN MODELS

SERVER-SIDE ACTIVITY DIAGRAM:

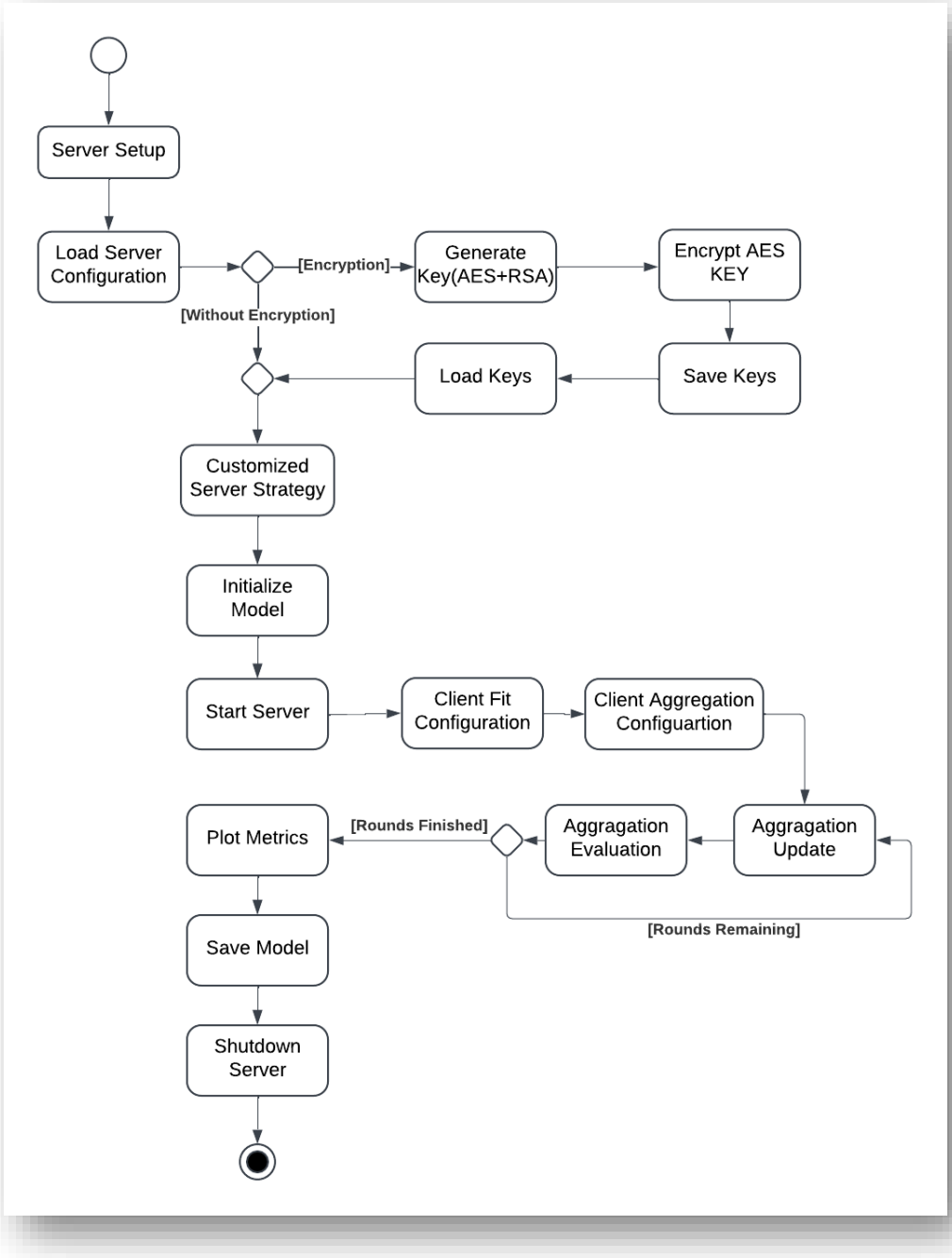


Figure 2 Server-Side Activity Flow

SERVER SETUP

1. **Load Server Configuration:** The server loads its configuration parameters from a YAML file, which includes details such as the encryption method, the number of rounds, the aggregation strategy, and other relevant settings.
2. **Generate Key (AES+RSA):** The server generates an AES key for data encryption and loads RSA keys for secure communication.
3. **Load Keys:** The server loads the generated AES and RSA keys into memory for use in encryption and decryption processes.
4. **Save Keys:** The server securely stores the generated keys for future use.
5. **Customized Server Strategy:** The server initializes a custom federated averaging strategy based on the loaded configuration, defining the communication protocol, data aggregation methods, and other necessary parameters.
6. **Initialize Model:** The server builds and loads the initial model based on the dataset requirements and the chosen strategy.
7. **Start Server:** The server begins listening for client connections, ready to handle incoming requests.
8. **Client Fit Configuration:** The server receives configuration parameters from clients, which include information about the client's data and model.
9. **Client Aggregation Configuration:** The server determines how to aggregate client data and model updates based on the received configurations.

TRAINING LOOP

1. **Aggregation Evaluation:** The server evaluates the aggregated results received from clients, assessing the effectiveness of the updates.
2. **Aggregation Update:** Based on the evaluation results, the server updates the model with the aggregated parameters.
3. **Plot Metrics:** The server plots relevant metrics based on the current state of the training process, providing visual feedback on performance.
4. **Save Model:** The server saves the current model state after each training round for backup or further analysis.
5. **Shutdown Server:** After all rounds of training are completed, the server gracefully shuts down.

CLIENT-SIDE ACTIVITY DIAGRAM:

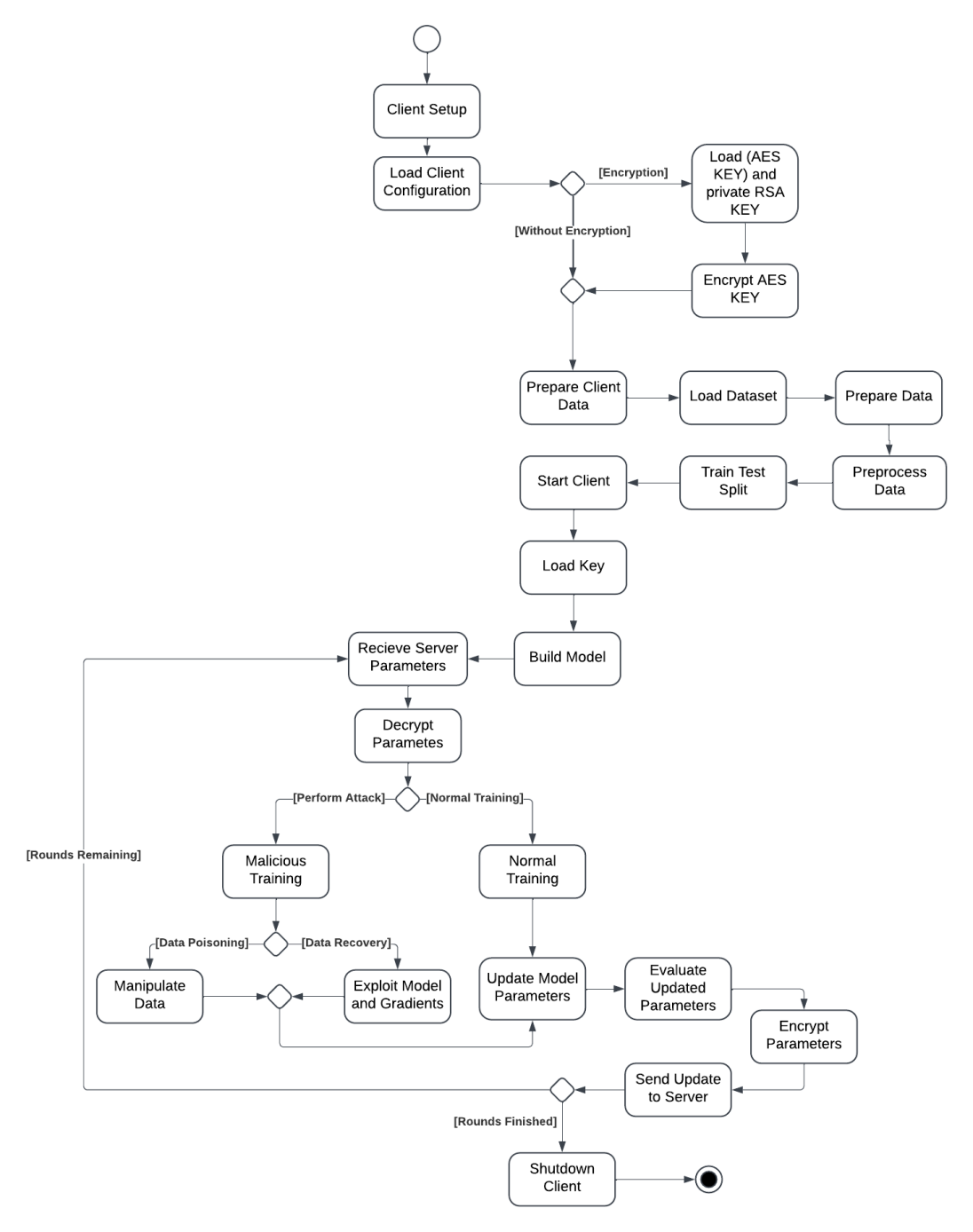


Figure 3 Client-Side Activity Flow

CLIENT SETUP:

1. **Load Client Configuration:** The client loads its configuration parameters from a YAML file, which includes data details, training settings, and connection information.
2. **Load (AES and private RSA) Key:** The client loads its AES key and private RSA key for secure communication with the server.
3. **Encrypt AES KEY:** The client encrypts its AES key using the server's public RSA key before sending it to the server.

CLIENT TRAINING

1. **Prepare Client Data:** The client prepares its local dataset for training by loading and preprocessing the data.
2. **Load Dataset:** The client loads the prepared dataset from local storage or a remote source.
3. **Prepare Data:** The client performs necessary preprocessing steps on the data, such as normalization or tokenization.
4. **Start Client:** The client initiates a connection to the server, establishing communication.
5. **Train Test Split:** The client splits the dataset into training and testing sets to evaluate model performance.
6. **Preprocess Data:** The client preprocesses the data to ensure it is in the correct format for training.
7. **Train Model:** The client trains its local model using the training dataset, adjusting parameters based on the data.
8. **Evaluate Model:** After training, the client evaluates the model's performance using the testing dataset to assess accuracy and loss.
9. **Send Model Update:** The client sends the updated model parameters back to the server for aggregation.
10. **Receive Aggregated Model:** The client receives the aggregated model from the server, which incorporates updates from all participating clients.
11. **Update Local Model:** The client updates its local model with the aggregated parameters received from the server.
12. **Repeat Training:** The client may repeat the training process for additional rounds, continuously improving its model based on the aggregated updates.

SHUTDOWN CLIENT

1. **Save Local Model:** The client saves its final model state for future use, ensuring that progress is not lost.

   Disconnect from Server: The client gracefully disconnects from the server, completing the training process and ending the session.

SEQUENCE DIAGRAM:

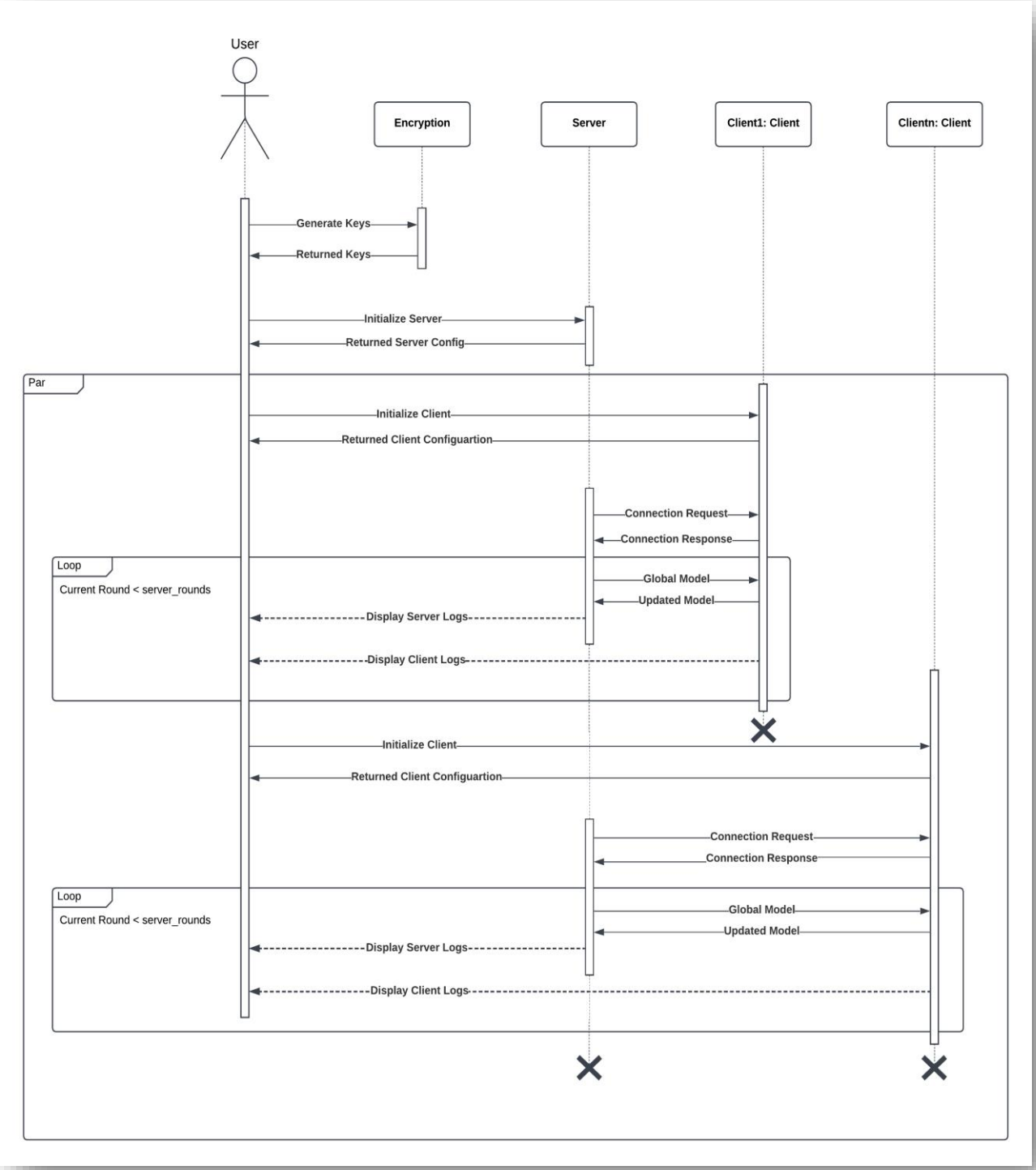


Figure 4 Sequence Diagram

DESCRIPTION:

The above sequence diagram outlines the interaction flow between different components in FedFusion. Below is the detailed description:

PARTICIPANTS (LIFELINES):

1. **User**: Represents the actor using the FedFusion platform.
2. **Encryption**: Class that handles the key generation and encryption and decryption processes.
3. **Server**: Manages initialization, configuration, and aggregation of model updates.
4. **Client1: Client**: Represents the first client node participating in federated setup.
5. **Clientn: Client**: Represents the N-TH client node (to indicate multiple clients).

KEY INTERACTIONS:

1. **User → Encryption**: The user initiates the process by requesting to "Generate Keys".
2. **Encryption → User**: Returns the generated keys.
3. **User → Server**: Sends a request to "Initialize Server".
4. **Server → User**: Returns the server configuration.
5. **User → Server**: Sends a request to "Initialize Client".
6. **Server → User**: Returns the client configuration.

PARALLEL CLIENT-SERVER INITIALIZATION:

The **"Par" (Parallel)** fragment indicates that multiple clients (Client1 and Clientn) are initialized in parallel:

1. **Client1**:
    a. **Client1 → Server**: Sends a "Connection Request".
    b. **Server → Client1**: Responds with a "Connection Response".
    c. **Server → Client1**: Sends the "Global Model".
    d. **Client1 → Server**: Sends back the "Updated Model".
2. **Clientn**: The same sequence of actions occurs for Clientn as for Client1.

LOOP: ITERATIVE TRAINING ROUNDS:

The **"Loop" fragment** specifies that the sequence repeats for a condition: Current Round < server_rounds. This represents **multiple rounds** of training for individual clients.

1. Inside Loop:
    a. **Server → Client1**: Sends the "Global Model".
    b. **Client1 → Server**: Sends back the "Updated Model".
    c. **User → Server**: Logs for the server are displayed (**Display Server Logs**).
    d. **User → Client1**: Logs for the client are displayed (**Display Client Logs**).

This process occurs for each training round, ensuring visibility into the system state and continuous training.

SYMBOLS AND ADDITIONAL DETAILS:

1. **Parallel (Par)**: The "Par" fragment is used to indicate concurrent operations between multiple clients with the server.
2. **Loop**: The "Loop" fragment specifies repetitive actions for training rounds until a condition is met.
3. **Termination (X)**: The large **"X"** symbols at the end of Client1 and Clientn lifelines indicate the termination of these processes. The client and server shutdown after the number of rounds have been met or the timeout has occurred.

CLASS DIAGRAM:

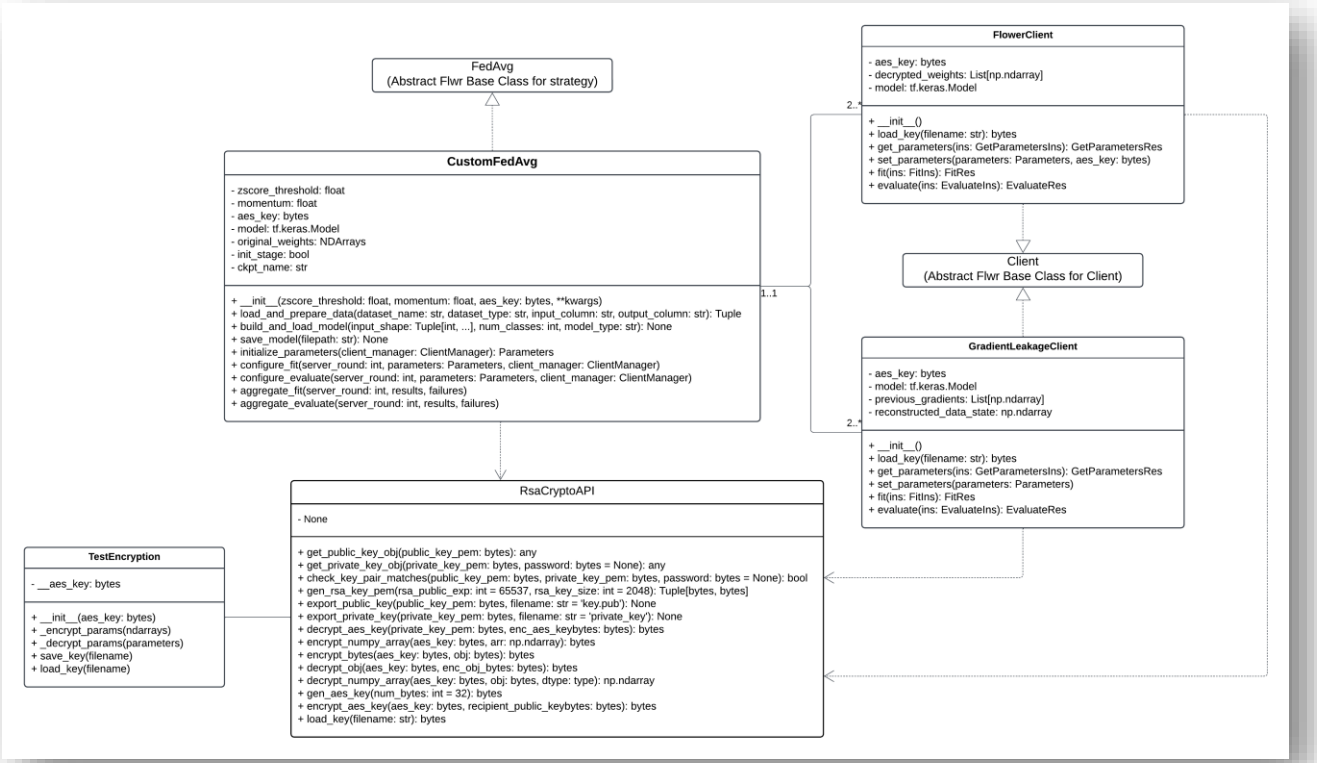


Figure 5 Class Diagram

DESCRIPTION:

FEDAVG (ABSTRACT BASE CLASS):

- **Role**: This is the foundational class for implementing the **Federated Averaging (FedAvg)** strategy in the Flower framework.
- **Purpose**: To act as a **template** or base class that defines the core logic for federated aggregation strategies. Custom implementations (like CustomFedAvg) extend this class to provide more tailored functionality.

- **Diagram Note**: It's depicted at the **top level** of the hierarchy. The dotted line indicates that CustomFedAvg **inherits** from FedAvg.

CUSTOMFEDAVG

- **Inheritance:** Extends FedAvg
- **Description**: This class customizes the FedAvg strategy by adding encryption, thresholding mechanisms, and custom client configuration logic.
- **Attributes:**
  - **Zscore_threshold** *(float)*: Threshold for standardization, anomaly detection, or outlier rejection (based on Z-score). Useful for data validation before client training.
  - **Momentum** *(float)*: Controls the momentum of the optimizer, aiding in faster convergence.
  - **Aes_key** *(bytes)*: AES encryption key used for encrypting and decrypting model weights.
  - **Model** *(tf.keras.Model)*: TensorFlow/Keras model used in the training pipeline.
  - **Original_weights** *(NDArray)*: Stores initial model weights for reference before any updates.
  - **Init_stage** *(bool)*: Tracks whether the server is in the initialization stage.
  - **Ckpt_name** *(str)*: Name for saving checkpoint files during the training rounds.
- **Methods:**
  - **__init__**: Initializes all attributes like zscore_threshold, momentum, and aes_key.
  - **Load_and_prepare_data**: Loads, processes, and prepares datasets for client training.
  - **Build_and_load_model**: Constructs and loads a TensorFlow/Keras model based on input parameters like shape and number of classes.
  - **Save_model**: Saves the model weights to a specified file path for checkpointing.
  - **Initialize_parameters**: Initializes parameters and shares them with clients at the start of the training rounds.
  - **Configure_fit / configure_evaluate**: Configures clients' fitting (training) and evaluation processes during a round of FL.
  - **Aggregate_fit / aggregate_evaluate**: Aggregates client updates (weights, losses) during the training/evaluation rounds.
- **Diagram Note:** It is an extended class to implement custom features
- **Composition**: CustomFedAvg interacts with **RsaCryptoAPI** to secure the encryption flow.

CLIENT (ABSTRACT BASE CLASS)

- **Role**: Base class for any FL client.
- **Purpose**: Provides a common interface that clients like FlowerClient and GradientLeakageClient inherit.
- **Diagram Note**: Positioned as an abstract class with solid lines indicating it is extended by FlowerClient and GradientLeakageClient.

FLOWERCLIENT

- **Inheritance:** Extends Client
- **Attributes**:
  - **Aes_key** *(bytes)*: AES encryption key for secure data exchange between client and server.
  - **Decrypted_weights** *(List[np.ndarray])*: Model weights after decryption.
  - **Model** *(tf.keras.Model)*: TensorFlow/Keras model for local training and evaluation.
- **Methods:**

  - **__init__**: Initializes encryption keys, weights, and the model.
  - **Load_key**: Loads encryption keys for secure communication.
  - **Get_parameters / set_parameters**: Retrieves and updates model parameters during FL rounds.
  - **Fit**: Performs local training (fit) of the model on the client-side data.
  - **Evaluate**: Evaluates the local model performance after training.
- **Purpose:** Securely decrypts model weights, trains locally, and returns securely encrypted updates to the server.
- **Diagram Note:** Solid lines show its inheritance from the abstract Client class.

GRADIENTLEAKAGECLIENT

- **Inheritance:** Extends Client
- **Attributes**:
  - **Aes_key** *(bytes)*: AES encryption key for data protection.
  - **Model** *(tf.keras.Model)*: Local TensorFlow/Keras model.
  - **Previous_gradients** *(List[np.ndarray])*: Gradients computed in the previous training round.
  - **Reconstructed_data_state** *(np.ndarray)*: Stores reconstructed data, likely for **gradient leakage research**.
- **Methods:** Similar to FlowerClient such as load_key, get_parameters, set_parameters, fit, evaluate.
- **Purpose:** Used to simulate gradient leakage scenarios for research purposes.

RSACRYPTOAPI

- **Role**: Provides encryption and decryption functionalities using RSA and AES algorithms.
- **Purpose**: Ensures secure encryption of model weights and keys before sharing between clients and servers.
- **Key Methods:**
  - **Get_public_key_obj / get_private_key_obj**: Loads public/private RSA keys.
  - **Encrypt_numpy_array / decrypt_numpy_array**: Encrypts and decrypts NumPy arrays, such as model weights.
  - **Encrypt_aes_key / decrypt_aes_key**: Encrypts and decrypts the AES keys for secure key sharing.
  - **Export_public_key / export_private_key**: Exports RSA keys to files for persistence.
  - **Load_key**: Loads keys from files.

TESTENCRYPTION

- **Purpose**: Utility class to test encryption functionalities.
- **Attributes:**
  - _aes_key *(bytes)*: Private AES key.
- **Methods:**
  - **_encrypt_params / _decrypt_params**: Tests encryption and decryption of model parameters.
  - **Save_key / load_key**: Saves and loads AES keys for testing.

DIAGRAM RELATIONSHIPS:

Inheritance:

- CustomFedAvg → FedAvg
- FlowerClient and GradientLeakageClient → Client.

Composition:

- CustomFedAvg depends on RsaCryptoAPI for encryption.

Interaction:

- CustomFedAvg interacts with FlowerClient and GradientLeakageClient for federated rounds.

Encryption:

- FlowerClient and GradientLeakageClient rely on RsaCryptoAPI for encrypting model weights.

DATA FLOW DIAGRAM:

LEVEL 1 DATA FLOW DIAGRAM DOCUMENTATION

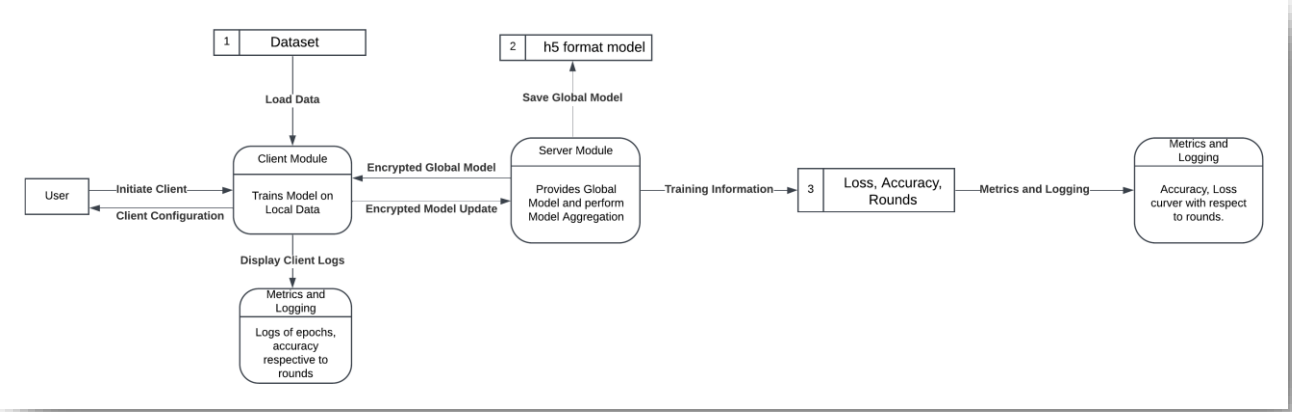


Figure 6 Contextual Data Flow Diagram

PURPOSE:

The Level 1 Data Flow Diagram (DFD) provides a **high-level overview** of the Federated Learning system. It captures the interactions between key entities involved in the system and describes the main processes.

ENTITIES:

User

- Initiates the federated learning process.
- Configures the client module with necessary parameters.

Dataset

- Supplies local training data to the client module.
- The data can be raw, normalized, or preprocessed based on user requirements.

Client Module

- Loads the local dataset for training.
- Trains a local model using the dataset.
- Sends **encrypted updates** (model parameters) to the server module for aggregation.

Server Module

- Provides the initial **global model** to clients for training.
- Aggregates encrypted local updates received from multiple clients.
- Produces an updated **global model** after aggregation.
- Stores the final global model for future rounds or evaluation.

Metrics Logging

- Logs key performance indicators (e.g., accuracy, loss, training rounds).
- Provides visualizations or reports for the user to analyze the training progress.

PROCESSES AND FLOWS:

User → Client Module

- The user initializes and configures the client module.
- Flow: Client Initialization and Training Configuration.

Dataset → Client Module

- The client module retrieves the local dataset for training.
- Flow: Load Local Dataset.

Client Module → Server Module

- The client module sends **encrypted model updates** to the server.
- Flow: Send Encrypted Updates.

Server Module → Client Module

- The server module distributes the **global model** to clients for the next training round.
- Flow: Distribute Global Model.

Server Module → Metrics Logging

- Logs metrics such as **accuracy** and **loss** after aggregation.
- Flow: Send Training Metrics.

Metrics Logging → User

- Displays performance metrics (accuracy, loss curves, training logs).
- Flow: Display Metrics.

DATA STORES:

- **Local Dataset**: Stored at the client module.
- **Global Model**: Stored at the server module in .h5 format.

LEVEL 2 DATA FLOW DIAGRAM DOCUMENTATION

PURPOSE:

The Level 2 Data Flow Diagram (DFD) provides a **detailed view** of the federated learning system. It includes processes like **encryption**, **preprocessing**, and **attack simulations** alongside the core client-server interactions.

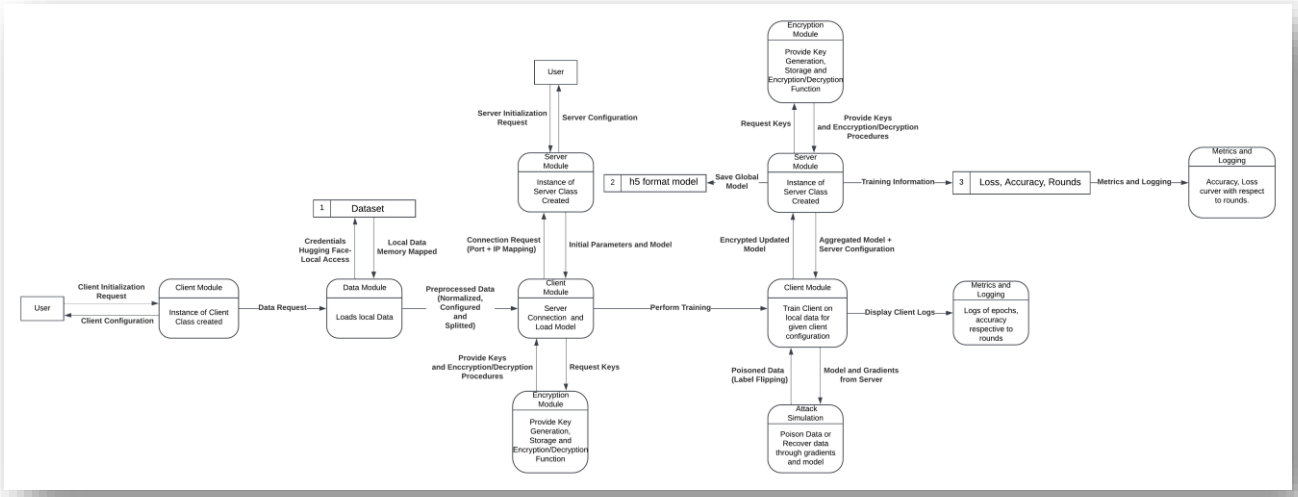


Figure 7 Detailed Data Flow Diagram

ENTITIES:

User

- Configures and initializes the system.
- Monitors the training progress and performance metrics.

Dataset Module

- Loads and preprocesses local data, including normalization and splitting into training/testing sets.

Client Module

- Loads the preprocessed local dataset.
- Trains a local model using the dataset.
- Requests **encryption keys** from the Encryption Module.
- Sends encrypted updates to the Server Module.

Encryption Module

- Generates encryption and decryption keys for secure communication.
- Encrypts the client's model updates before transmission.

Server Module

- Provides the **initial global model** to clients.
- Aggregates encrypted updates received from clients.
- Stores the aggregated model securely in .h5 format.

Attack Simulation

- Simulates adversarial attacks like **data poisoning** (e.g., label flipping).
- Conducts gradient recovery experiments to test system vulnerabilities.

Metrics Logging

- Logs training metrics such as **accuracy, loss, and round information**.
- Provides visualizations and reports for performance analysis.

PROCESSES AND FLOWS:

User → Client Module

- The user initializes the federated learning process and configures clients.
- Flow: Initialize Client, Configure Training.

Dataset Module → Client Module

- Provides preprocessed data (e.g., normalized, split data).
- Flow: Load and Preprocess Data.

Client Module → Encryption Module

- Requests encryption keys to secure model updates.
- Flow: Request Encryption Keys.

Client Module → Server Module

- Sends **encrypted updates** (trained model parameters).
- Flow: Send Encrypted Updates.

Server Module → Encryption Module

- Ensures updates are encrypted/decrypted before aggregation.
- Flow: Encrypt/Decrypt Updates.

Server Module → Client Module

- Sends the **global model** back to clients for the next training round.
- Flow: Distribute Global Model.

Attack Simulation → Client Module

- Injects adversarial attacks into the local training process.
- Flow: Simulate Label Flipping or Data Poisoning.

Server Module → Metrics Logging

- Sends aggregated performance metrics.
- Flow: Log Training Metrics.

Metrics Logging → User

- Displays key performance metrics for analysis.
- Flow: Display Logs and Visualizations.

DATA STORES:

Local Dataset:

- Preprocessed data stored at the client module for training.

Encryption Keys:

- Stored securely in the Encryption Module.

Global Model:

- Aggregated global model stored at the server module in .h5 format.

Metrics Logs:

- Logs stored for accuracy, loss, and training rounds in the Metrics Logging module.

Level 1 provides a high-level overview of the federated learning system. It focuses on the flow between **User, Dataset, Client, Server**, and **Metrics Logging** modules. **Level 2** expands upon Level 1 by introducing **encryption mechanisms**, **preprocessing**, and **attack simulations** while detailing the data flows between all modules.

BOX AND LINE DIAGRAM:

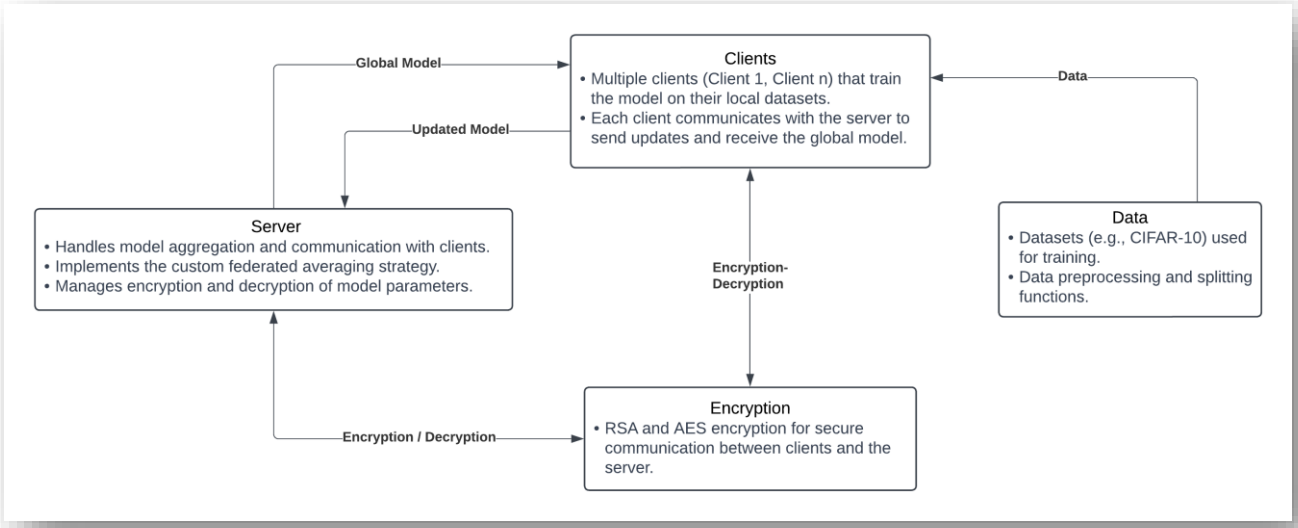


Figure 8 Box and Line Diagram

CLIENTS

Represented as multiple clients (e.g., Client 1, Client 2, ..., Client n).
Role:

- Train the model on **local datasets** independently.
- Communicate with the **Server** by sending model updates and receiving the global model.
- This ensures data privacy because data remains on the client side and only model parameters are exchanged.

Interaction:

- Clients receive the **Global Model** from the server.
- After training, they send the **Updated Model** back to the server.

2. SERVER

The server acts as the central aggregator and coordinator of the federated learning process.
Responsibilities:

Model Aggregation:
- Collects the updated models from all clients.
- Implements the **Federated Averaging Strategy (FedAvg)** or other aggregation techniques to update the global model.

Communication:
- Facilitates secure exchange of the model parameters between clients.

Encryption Management:
- Handles encryption and decryption of model parameters to ensure data security.

Interaction:
- Sends the **Global Model** to clients.
- Receives **Updated Models** from clients and aggregates them.

DATA

- Refers to datasets used for training (e.g., **CIFAR-10**).
- This data is **local** to each client and is never shared directly with the server.
- Includes **data preprocessing and splitting functions** to prepare datasets for training.
- **Interaction**:
    - The data remains within the client systems and is used locally for training the models.

ENCRYPTION

- Ensures secure communication between the server and the clients. Uses cryptographic techniques such as **RSA** and **AES encryption**.
- **Key Features**:
    - **RSA** (Asymmetric Encryption): Often used to securely share symmetric encryption keys.
    - **AES** (Symmetric Encryption): Used for encrypting and decrypting model updates and parameters.
- **Interaction**:
    - Encryption/Decryption happens between the **Server** and **Clients** to protect the model updates.

KEY FLOW OF COMMUNICATION:

1. The **Server** sends the initial **Global Model** to all **Clients**.
2. Each **Client** trains the model on its local **Data** and generates an **Updated Model**.
3. The **Updated Model** is encrypted (using RSA/AES) and sent back to the **Server**.
4. The **Server** decrypts, aggregates the models (FedAvg), and updates the **Global Model**.
5. This updated global model is redistributed to the clients for the next training round.

DATA DESIGN

INFORMATION DOMAIN TRANSFORMATION INTO DATA STRUCTURES

In the FedFusion, the information domain is transformed into data structures that represent the various entities involved in the federated learning process. These entities include clients, models, datasets, and communication parameters. The data structures are designed to facilitate efficient storage, processing, and organization of the information required for the system's functionality.

MAJOR DATA ENTITIES AND THEIR STRUCTURES

CLIENT:

- **Data Structure:** Each client is represented as an object with attributes such as client_id, local_data, model_weights, and metrics.
- **Storage:** Client data is stored in memory during runtime, and model weights can be serialized and saved to disk if needed utilizing pickle files for serialized object storage.
- **Processing:** Clients process their local data to train models and send updates to the server.

SERVER:

- **Data Structure:** The server maintains a global model represented by attributes such as global_model_weights, client_updates, and metrics.
- **Storage:** The global model weights can be stored in memory or saved to disk for persistence.
- **Processing:** The server aggregates updates from clients and manages the global model.

MODEL:

- **Data Structure:** Models are defined using classes that encapsulate attributes like input_shape, num_classes, and architecture.
- **Storage:** Model definitions and weights can be serialized and stored in files (e.g., HDF5 format for Keras models).
- **Processing:** Models are trained using data provided by clients and updated based on aggregated client updates.

DATASET:

- **Data Structure:** Datasets are represented as collections of data points, each containing features and labels. This can be structured as a list of dictionaries or a NumPy array.
- **Storage:** Datasets can be loaded from external sources (e.g., CSV files, Hugging Face datasets) and stored in memory during processing.
- **Processing:** Datasets are preprocessed (e.g., normalization, tokenization) before being used for training.

COMMUNICATION PARAMETERS:

- **Data Structure:** Communication parameters include attributes such as encrypted_model_weights, aes_key, and client_id.

- **Storage:** These parameters are typically stored in memory during communication and can be serialized for secure transmission.
- **Processing:** Parameters are encrypted before being sent to the server and decrypted upon receipt.

DATA REPRESENTATION DIAGRAMS

ERD

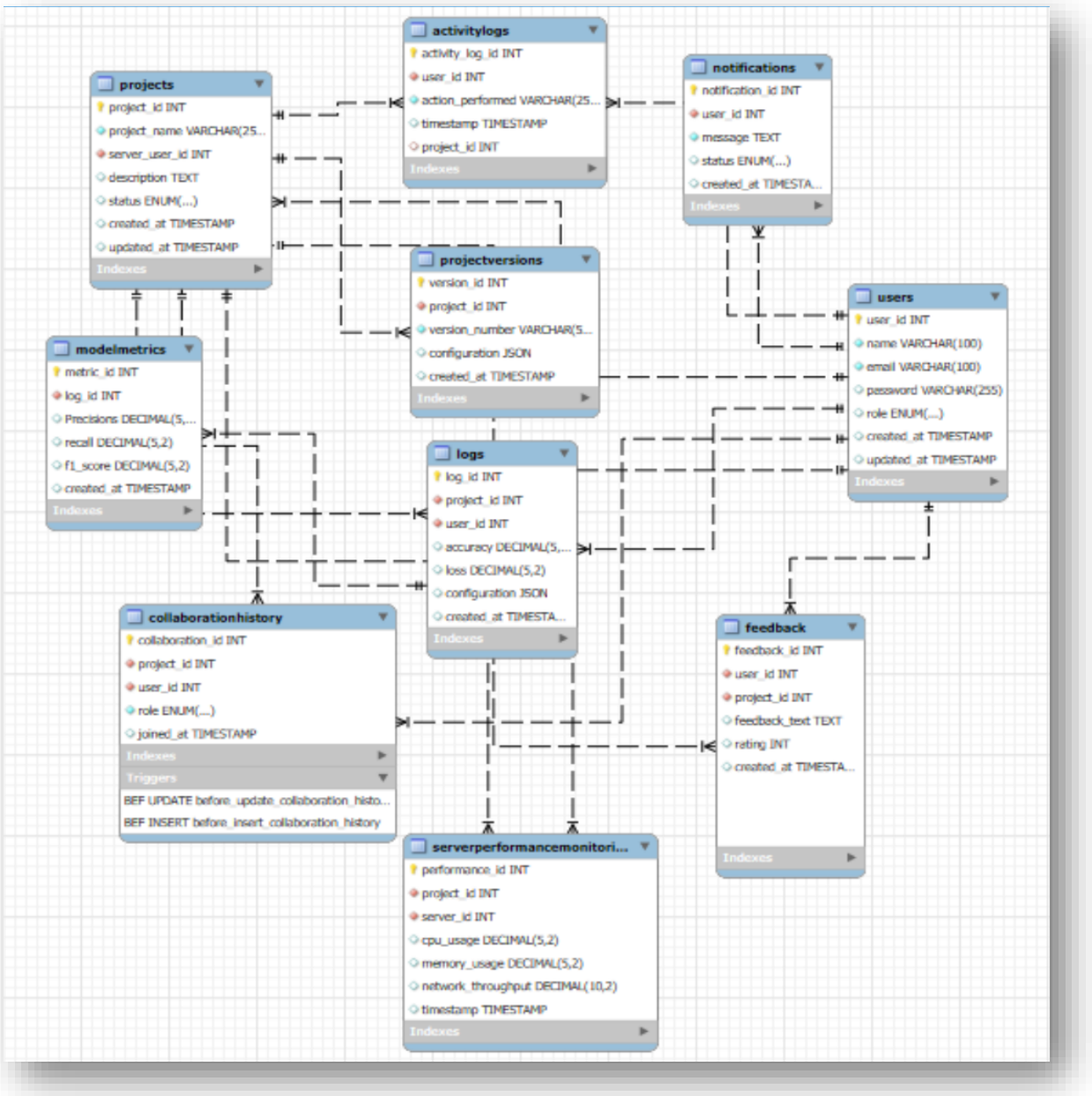


Figure 9 Front-End ERD

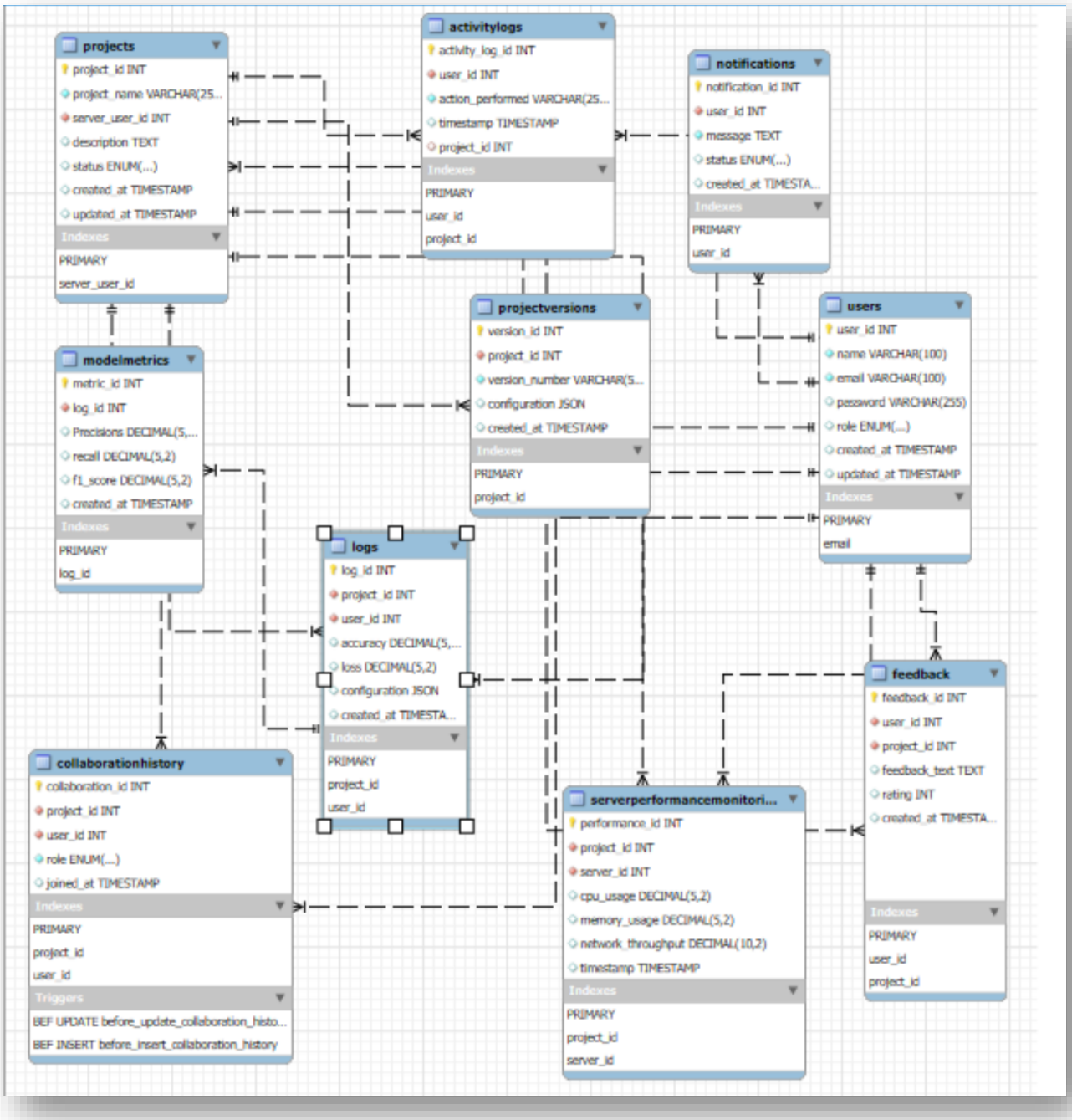


Figure 10 Front-End ERD with Indexing

RELATIONSHIPS AND MAPPING FUNCTIONS

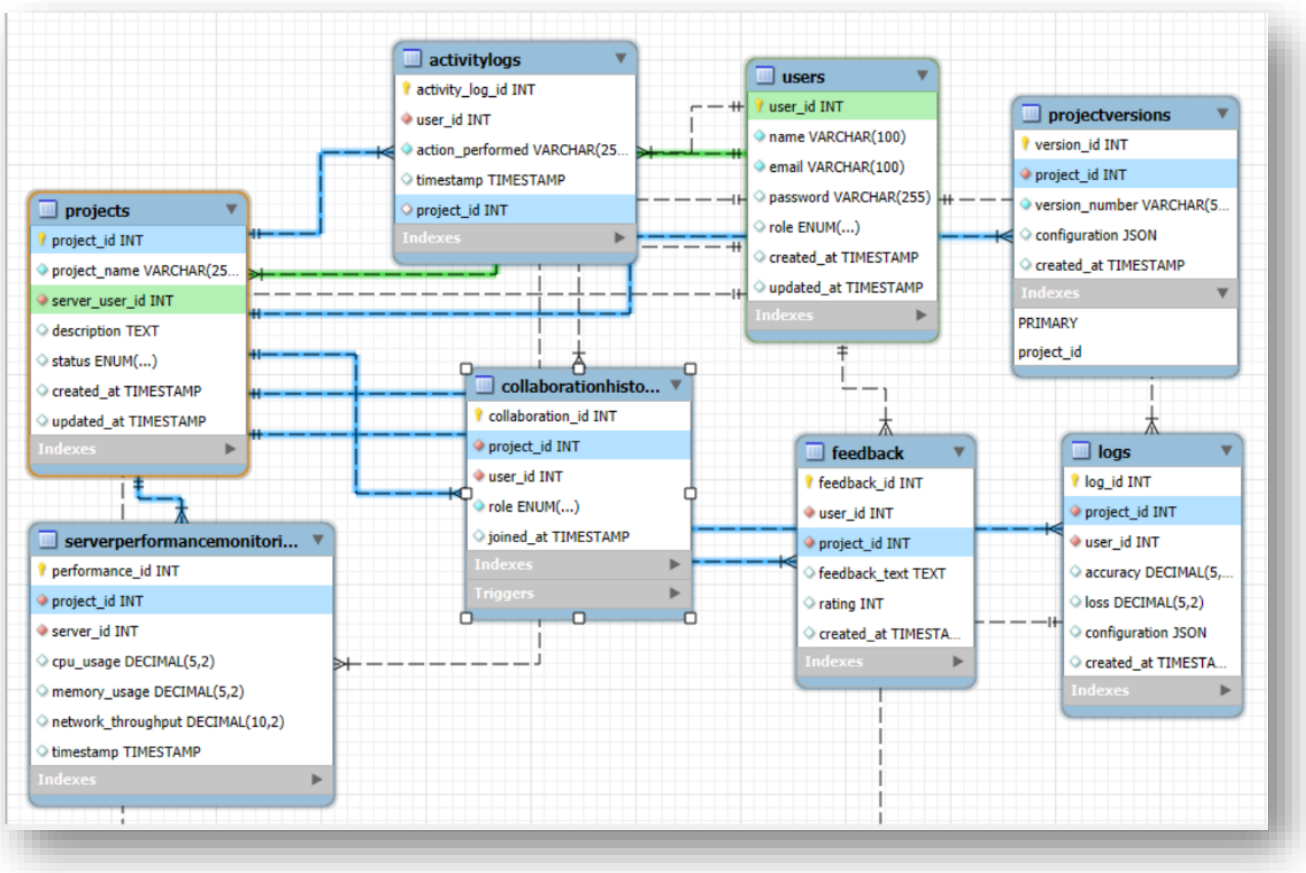


Figure 11 User and Project Relation

Users ↔ Projects:

Relationship Type: One-to-Many

Mapping Function:

- One user can create or join multiple projects.
- One project has one server user (defined by the server_user_id foreign key) and multiple collaborators (stored in the collaboration history table).
- The server_user_id foreign key in the Projects table references the user_id primary key in the Users table.

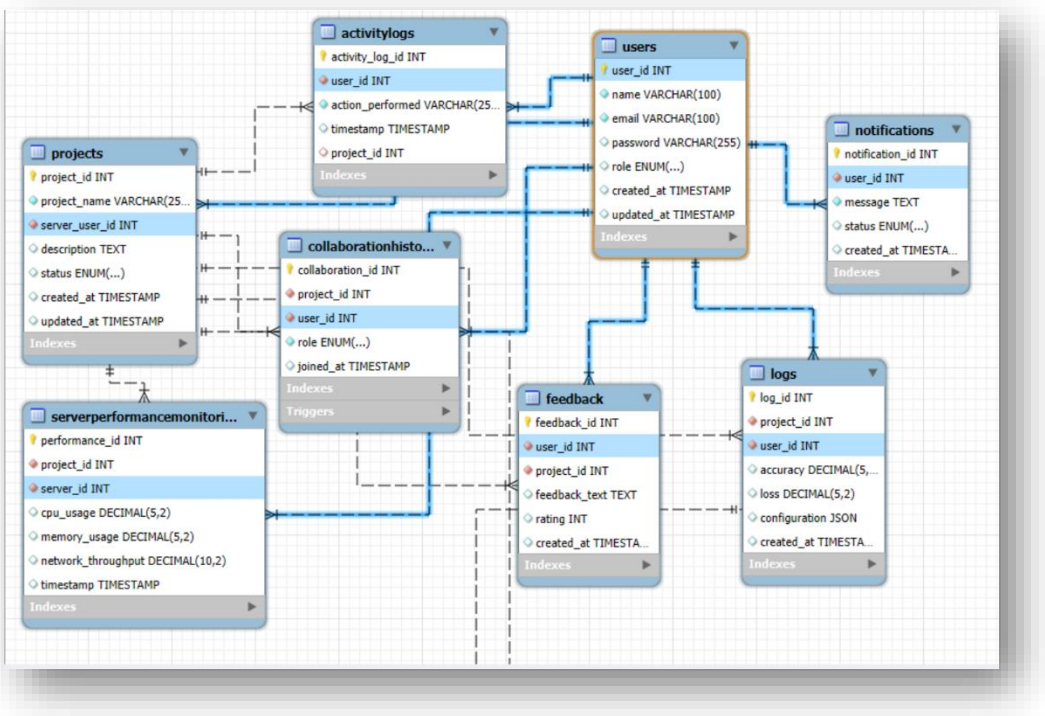


Figure 12 All User Relations

Users ↔ Collaboration History:

Relationship Type: Many-to-Many

Mapping Function:

- One user can participate in multiple projects with different roles (server or client).
- One project can have multiple users participating with different roles.
- The user_id foreign key in the collaboration history table references the user_id primary key in the Users table.
- The project_id foreign key in the collaboration history table references the project_id primary key in the Projects table.

Users ↔ Feedback:

Relationship Type: One-to-Many

Mapping Function:

- One user can provide feedback for multiple projects.
- One feedback entry belongs to a single user and project.
- The user_id foreign key in the Feedback table references the user_id primary key in the Users table.
- The project_id foreign key in the Feedback table references the project_id primary key in the Projects table.
- The rating column in the Feedback table has a CHECK constraint to ensure the value is between 1 and 5.

Users ↔ Activity Logs:

Relationship Type: One-to-Many

Mapping Function:

- One user can have multiple activity log entries.
- One activity log entry belongs to a single user.
- The user_id foreign key in the activity logs table references the user_id primary key in the Users table.
- The project_id foreign key in the activity logs table references the project_id primary key in the Projects table.

Users ↔ Notifications:

Relationship Type: One-to-Many

Mapping Function:

- One user can receive multiple notifications.
- One notification belongs to a single user.
- The user_id foreign key in the Notifications table references the user_id primary key in the Users table.

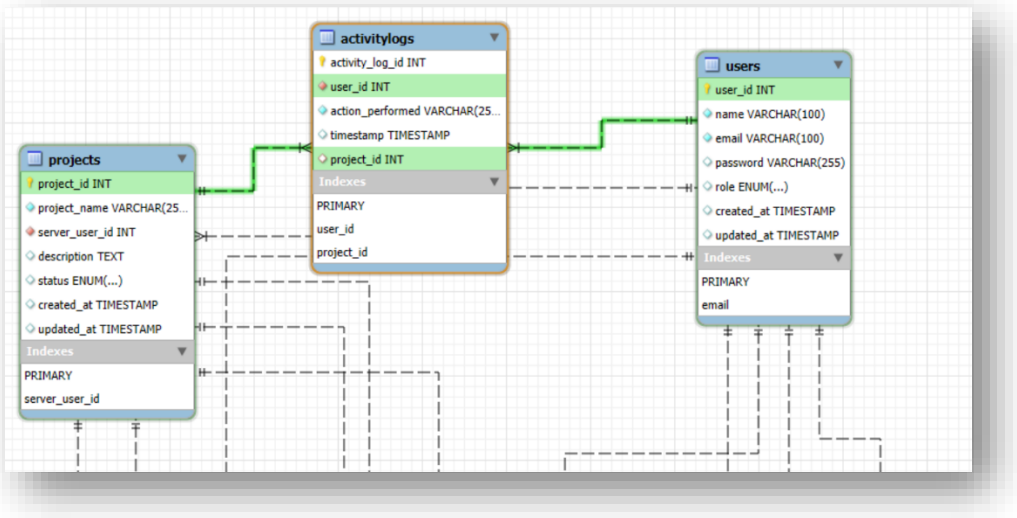


Figure 13 Project and Activity Relation

Projects ↔ Logs:

Relationship Type: One-to-Many

Mapping Function:

- One project can have multiple logs associated with it.
- One log belongs to a single project.
- The project_id foreign key in the Logs table references the project_id primary key in the Projects table.
- The user_id foreign key in the Logs table references the user_id primary key in the Users table.

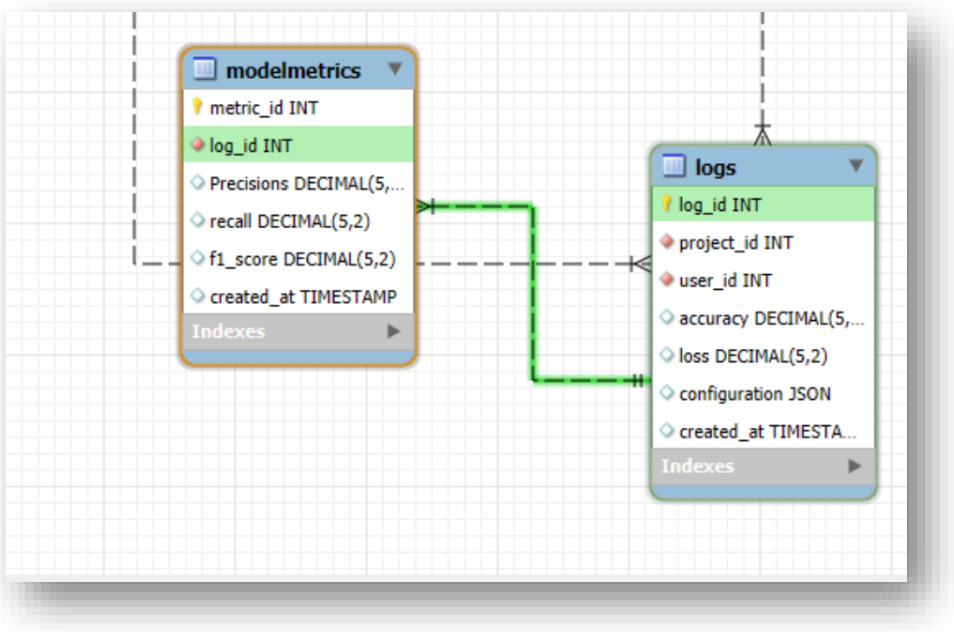


Figure 14 Metrics and Logging Relation

Logs ↔ Model Metrics:

Relationship Type: One-to-One

Mapping Function:

- One log can have one set of model metrics associated with it.
- One set of model metrics belongs to a single log.
- The log_id foreign key in the model metrics table references the log_id primary key in the Logs table.

Projects ↔ Project Versions:

Relationship Type: One-to-Many

Mapping Function:

- One project can have multiple versions.
- One version belongs to a single project.
- The project_id foreign key in the project versions table references the project_id primary key in the Projects table.
- The project_id and version_number columns in the project versions table have a unique constraint to ensure version uniqueness per project.

Projects ↔ Server Performance Monitoring:

Relationship Type: One-to-Many

Mapping Function:

- One project can have multiple server performance monitoring entries.
- One server performance monitoring entry belongs to a single project.
- The project_id foreign key in the server performance monitoring table references the project_id primary key in the Projects table.
- The server_id foreign key in the server performance monitoring table references the user_id primary key in the Users table.

CONSTRAINTS AND BUSINESS LOGIC

1. **Server-Client Relationship**:

   - Constraint: A project must have exactly one server user.
   - Business Logic:
     - When inserting or updating a collaboration history record, the system should ensure that there is only one record with the role set to 'server' for the given project_id.
     - This is implemented using the provided triggers before_insert_collaboration_history and before_update_collaboration_history.

2. **Logs**:

   - Constraint: Logs are immutable once created.
   - Business Logic:
     - The system should not allow updates or deletions on the Logs table.

3. **Versioning**:

   - Constraint: Each version must be unique for the project.
   - Business Logic:
     - When inserting a new project version record, the system should check if the combination of project_id and version_number already exists and raise an error if so.
     - This is implemented using the UNIQUE constraint on the project_id and version_number columns in the project versions table.

4. **Notifications**:

   - Constraint: Notifications must be tied to users and include timestamps.
   - Business Logic:
     - When inserting a new Notification record, the system should ensure that the user_id foreign key references a valid user, and the created_at timestamp is set correctly.

5. **Feedback**:

   - Constraint: Feedback rating must be between 1 and 5.
   - Business Logic:
     - When inserting a new Feedback record, the system should ensure that the rating value is between 1 and 5 (inclusive).
     - This is implemented using the CHECK constraint on the rating column in the Feedback table.

CONCEPT BEHIND ERD

COMPREHENSIVE LIST OF ENTITIES AND ATTRIBUTES

1. Users Table

- **Purpose**: Store user information and roles.
- **Attributes**:
  - User ID (Primary Key)
  - Name
  - Email
  - Password (if not using Firebase)
  - Role (e.g., admin, regular user)
  - Created At (timestamp)
  - Updated At (timestamp)

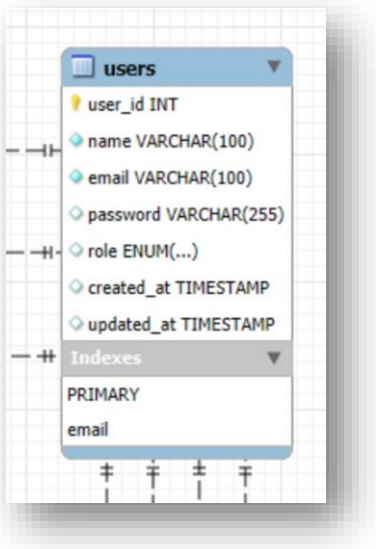


Figure 15 User Table

2. Projects Table

- **Purpose**: Define projects with hierarchical support (server and clients).
- **Attributes**:
  - Project ID (Primary Key)
  - Project Name
  - Server User ID (Foreign Key to Users)
  - Description
  - Status (e.g., Active, Completed)
  - Created At (timestamp)
  - Updated At (timestamp)

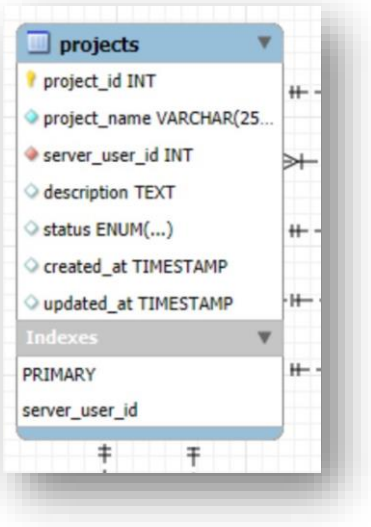


Figure 16 Project Table

3. Collaboration History Table

- **Purpose**: Maps user roles in projects (server and client only).
- **Attributes**:
  - Collaboration ID (Primary Key)
  - Project ID (Foreign Key to Projects)
  - User ID (Foreign Key to Users)
  - Role (e.g., Server, Client)
  - Joined At (timestamp)
- **Business Rule**: Ensure a project has exactly one server and clients cannot join without the server.

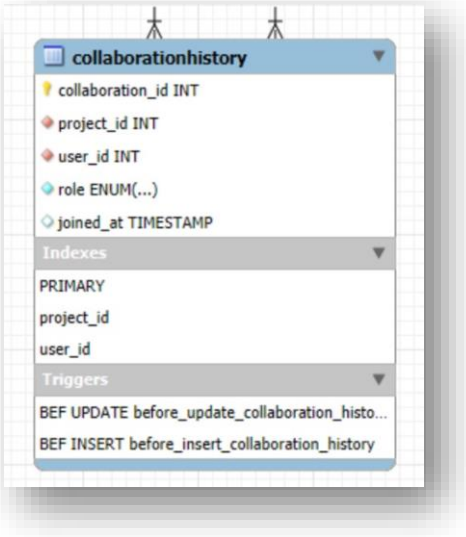


Figure 17 Collaboration History Table

4. Logs Table

- **Purpose**: Stores model configurations and results.
- **Attributes**:
  - Log ID (Primary Key)
  - Project ID (Foreign Key to Projects)
  - User ID (Foreign Key to Users)
  - Accuracy
  - Loss
  - Configuration (JSON for hyperparameters, etc.)
  - Created At (timestamp)

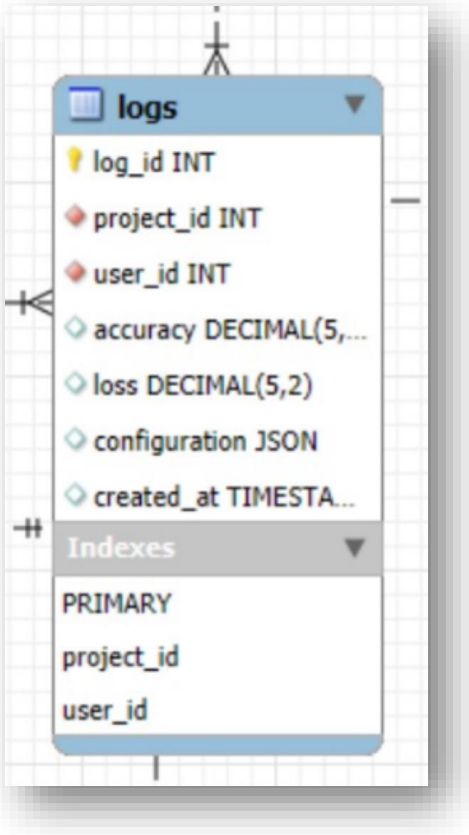


Figure 18 Logs Table

5. Activity Logs Table

- **Purpose**: Tracks user actions like logins, updates, and edits.
- **Attributes**:
  - Activity Log ID (Primary Key)
  - User ID (Foreign Key to Users)
  - Action Performed (e.g., “Joined Project,” “Updated Config”)
  - Timestamp
  - Project ID (Foreign Key: optional)

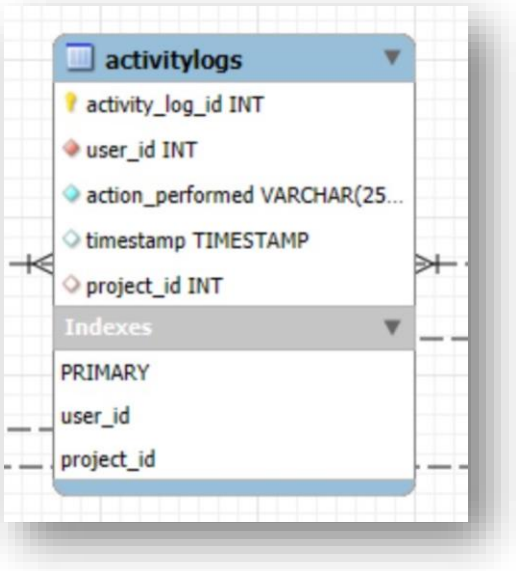


Figure 19 Activity Logs Table

6. Notifications Table

- **Purpose**: Send alerts to users for updates or collaboration invites.
- **Attributes**:
  - Notification ID (Primary Key)
  - User ID (Foreign Key to Users)
  - Message
  - Status (e.g., Unread, Read)
  - Created At (timestamp)

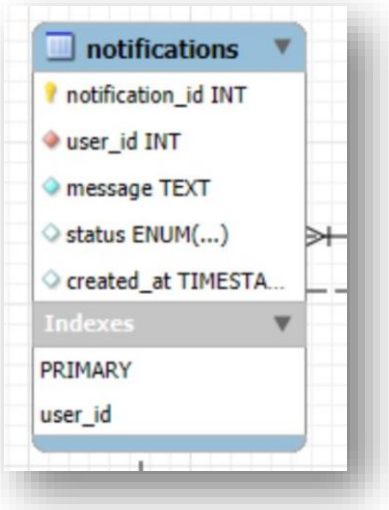


Figure 20 Notification Table

7. Project Versions Table

- **Purpose**: Manage version control for project configurations.
- **Attributes**:
  - Version ID (Primary Key)
  - Project ID (Foreign Key to Projects)
  - Version Number
  - Configuration (JSON)
  - Created At (timestamp)
- **Business Rule**: Enforce unique version numbers per project.

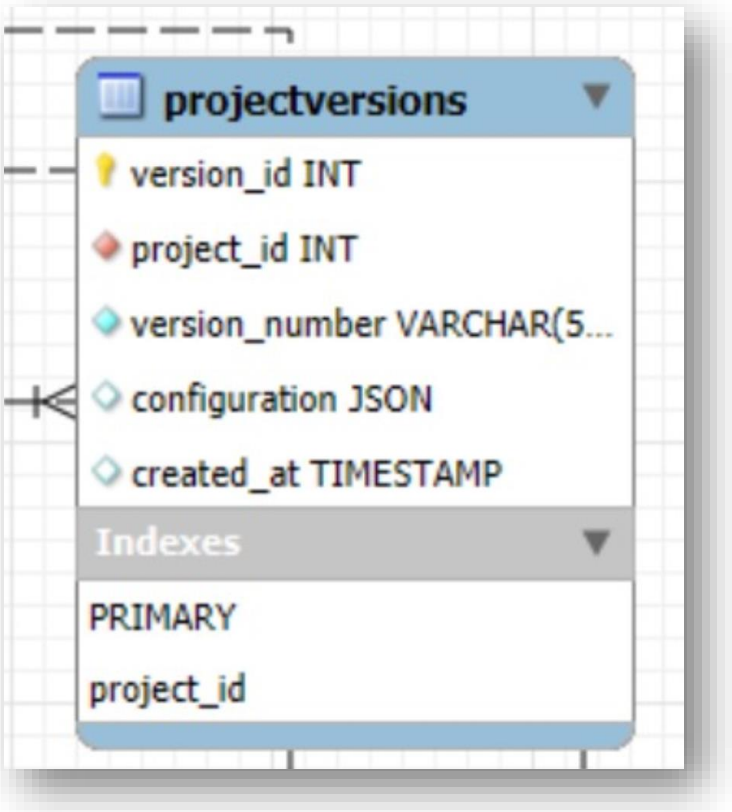


Figure 21 Project Versions Table

8. Model Metrics Table

- **Purpose**: Store detailed performance metrics like precision and recall.
- **Attributes**:
  - Metric ID (Primary Key)
  - Log ID (Foreign Key to Logs)
  - Precision
  - Recall
  - F1 Score
  - Created At (timestamp)

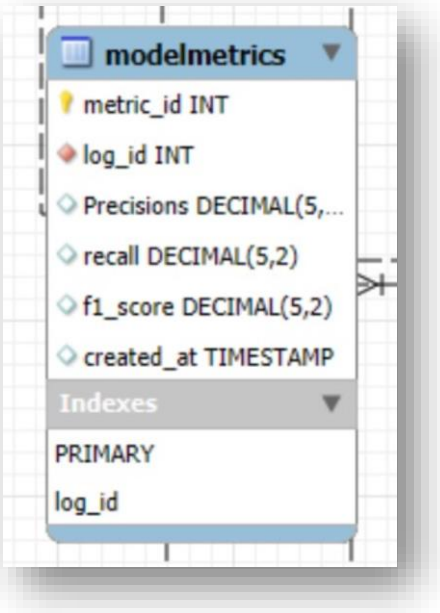


Figure 22 Model Metrics Table

9. Server Performance Monitoring Table

- **Purpose**: Track server stats during active projects.
- **Attributes**:
  - Performance ID (Primary Key)
  - Project ID (Foreign Key to Projects)
  - Server ID (Foreign Key to Users)
  - CPU Usage (%)
  - Memory Usage (%)
  - Network Throughput (Mbps)
  - Timestamp

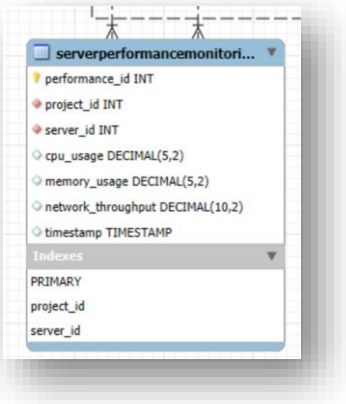


Figure 23 Server Performance Monitoring Table

10. Feedback Table

- **Purpose**: Collect user reviews or feedback about projects.
- **Attributes**:
  - Feedback ID (Primary Key)
  - User ID (Foreign Key to Users)
  - Project ID (Foreign Key to Projects)
  - Feedback Text
  - Rating (e.g., 1-5)
  - Created At (timestamp)

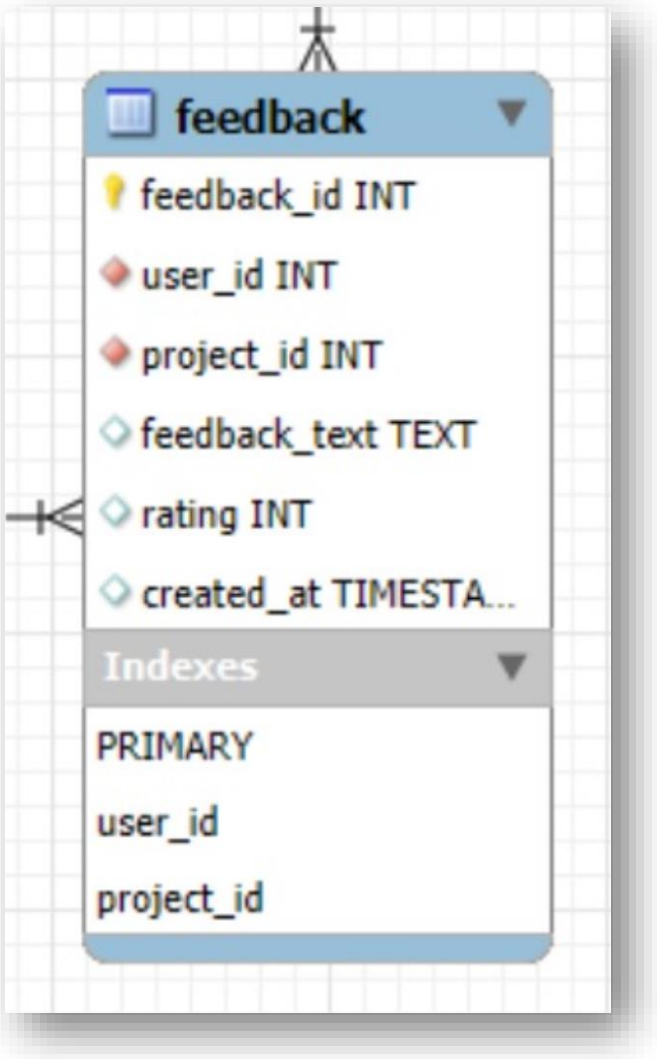


Figure 24 Feedback Table

RELATIONSHIPS

1. **Users ↔ Projects**: One user can create or join multiple projects; one project has one server and many collaborators.
2. **Users ↔ Collaboration History**: Maps user roles (server, client) to specific projects.
3. **Projects ↔ Logs**: Logs are tied to specific projects.
4. **Logs ↔ Model Metrics**: Detailed metrics for training sessions are linked to logs.
5. **Users ↔ Activity Logs**: Tracks all user actions.
6. **Users ↔ Notifications**: Alerts sent to users for specific events.
7. **Projects ↔ Project Versions**: Tracks different configuration versions.
8. **Projects ↔ Server Performance Monitoring**: Server stats logged for active projects.
9. **Projects ↔ Feedback**: Collects feedback for completed projects.

RULES AND CONSTRAINTS

1. **Server-Client Relationship**:
   - A project must have **exactly one server**.
   - A client cannot join without a server being assigned.
2. **Logs**:
   - Logs are immutable once created.
3. **Versioning**:
   - Each version must be unique for the project.
4. **Notifications**:
   - Notifications must be tied to users and include timestamps.

BACKEND DATA DICTIONARY

Alphabetical List of System Entities and major Data Entities in the FedFusion, along with their types and descriptions:

CLIENT

- Type: Object
- Description: Represents an individual participant in the federated learning system that trains a model on its local dataset.
- Attributes:
  - Client_id: Unique identifier for the client.
  - Local_data: The dataset used for training by the client.
  - Model_weights: The weights of the model trained by the client.
  - Metrics: Performance metrics (e.g., accuracy, loss) of the client's model.

COMMUNICATION

- Type: Object
- Description: Represents the parameters used for secure communication between clients and the server.
- Attributes:
  - Comm_id: Unique identifier for the communication instance.
  - Encrypted_weights: The encrypted model weights sent to the server.
  - Aes_key: The AES key used for encryption.
  - Client_id: Foreign key referencing the associated client.

DATASET

- Type: Object
- Description: Represents the datasets used for training and testing in the federated learning system.
- Attributes:
  - Dataset_id: Unique identifier for the dataset.
  - Features: The input features of the dataset.
  - Labels: The corresponding labels for the dataset.

MODEL

- Type: Object
- Description: Represents the machine learning models used by clients and the server.
- Attributes:
  - Model_id: Unique identifier for the model.
  - Input_shape: The shape of the input data for the model.
  - Num_classes: The number of output classes for classification tasks.
  - Architecture: The architecture of the model (e.g., dense, convolutional).

SERVER

- Type: Object
- Description: Represents the central server that coordinates the federated learning process.
- Attributes:
  - Server_id: Unique identifier for the server.
  - Global_model: The current state of the global model.
  - Metrics: Performance metrics (e.g., accuracy, loss) of the global model.

USER INTERFACE DESIGN

The FEDFUSION platform enables developers and enterprises to collaborate securely using federated learning techniques. This section provides an in-depth view of the user interface, explaining how the system functions from the user's perspective. It details screens, objects, user actions, navigation flows, and feedback mechanisms that ensure a smooth user experience.

FRONT END ACTIVITY DIAGRAM

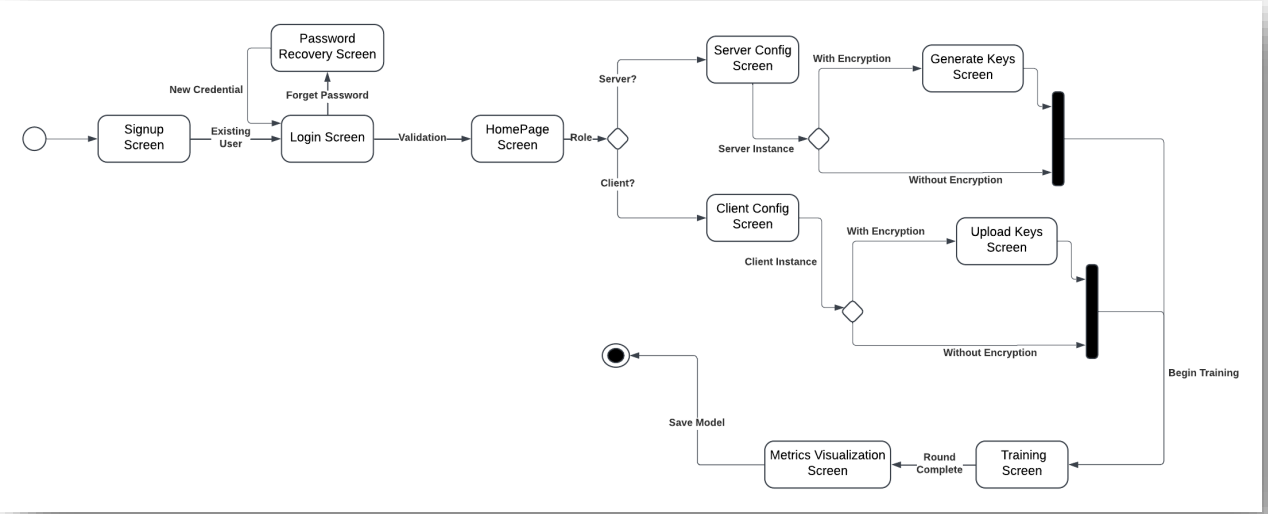


Figure 25 Frontend Flow

DESCRIPTION

1. **Sign-Up Screen:** Users create a new account with required credentials (email, username, password).
2. **Login Screen:** Existing users log in or navigate to password recovery if needed.
3. **Password Recovery Screen:** Allows users to reset their password by sending a recovery email.
4. **Home Page Screen:** Acts as the central hub where users select their role (Client or Server).
5. **Role Selection (Decision):** Users choose to either contribute data (Client) or host the collaboration (Server).
6. **Server Config Screen:** Server sets collaboration parameters like batch size, epochs, and model type.
7. **Client Config Screen:** Client provides server address and dataset path for participation.

8. **Data Privacy (Decision):** Server chooses between Encryption Mode or Open Mode for training.
9. **Generate Keys Screen:** Encryption keys (RSA and AES) are uploaded or generated for secure data handling.
10. **Training Screen:** Begins training with real-time logs and round updates.
11. **Metrics Visualization Screen:** Displays dynamic graphs of accuracy and loss during training.
12. **Save Model (End State):** Completes the process by saving the trained model.

LOGIN AND REGISTRATION SCREENS

Purpose: Authenticate users and provide access to the platform.
Features:

- Login options: EMAIL/PASSWORD, GOOGLE, TWITTER.
- Error validation for incorrect credentials.
- Password recovery flow.

Registration with password rules:

- Use 8 or more characters.
- Include one uppercase letter, one number, one lowercase letter, and one special character.

Feedback Messages:

- “Invalid credentials. Please try again.”
- “Password reset link sent successfully.”

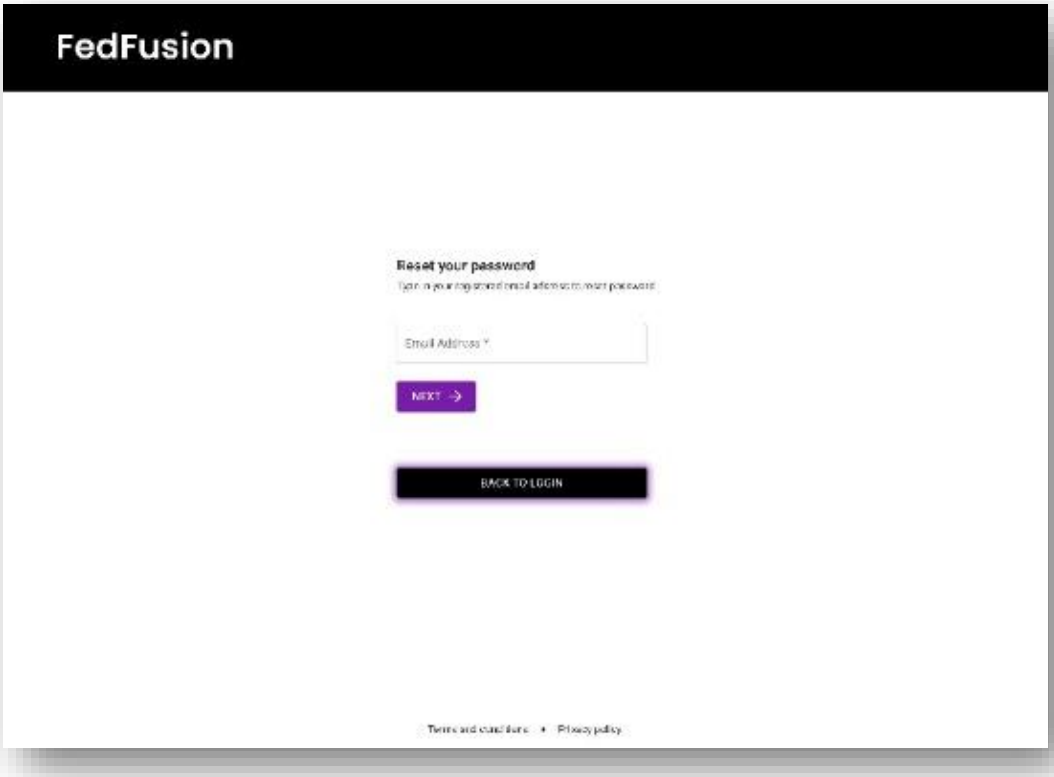


Figure 26 Login Screen 1

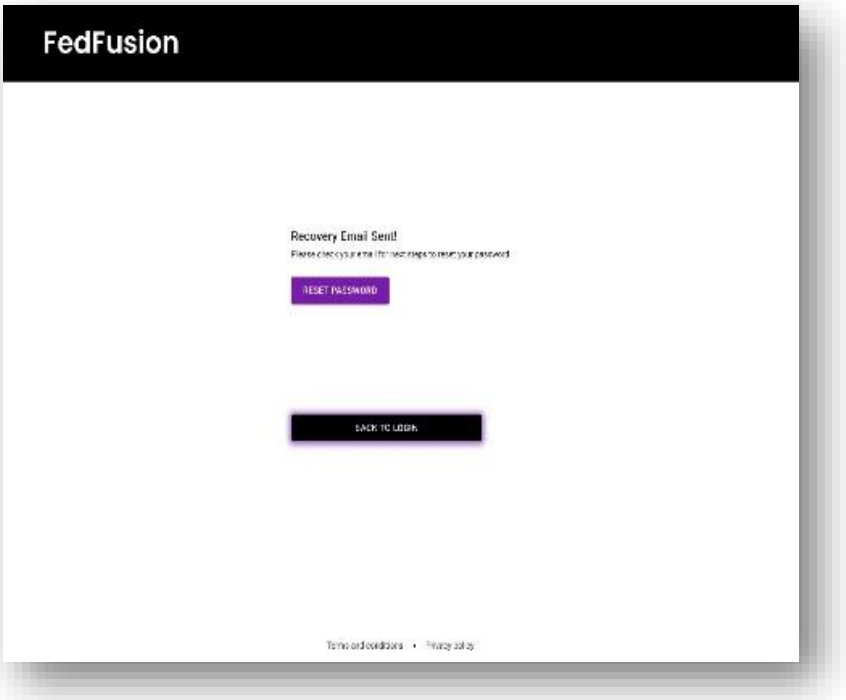


Figure 27 Login Screen 2

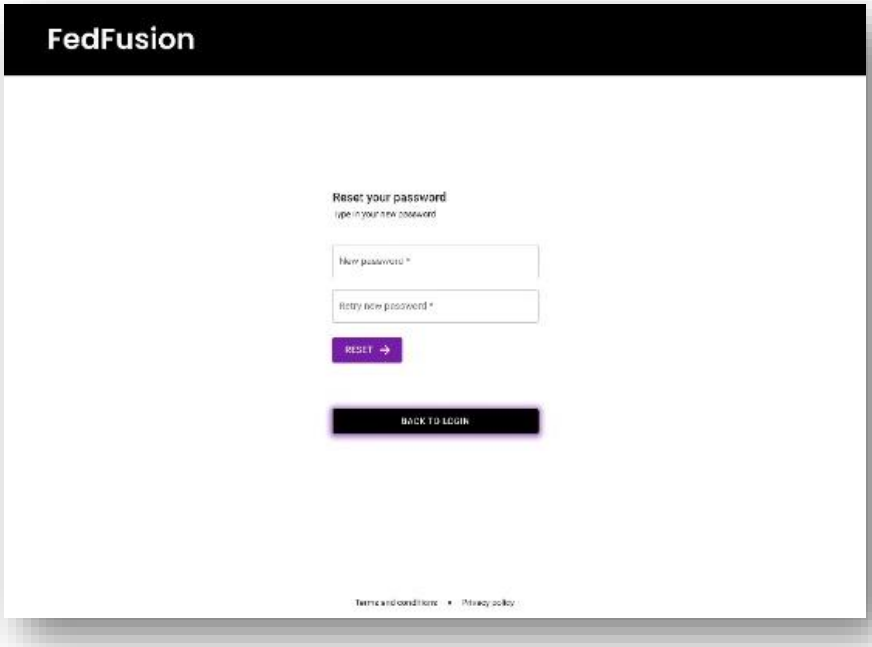


Figure 28 Login Screen 3

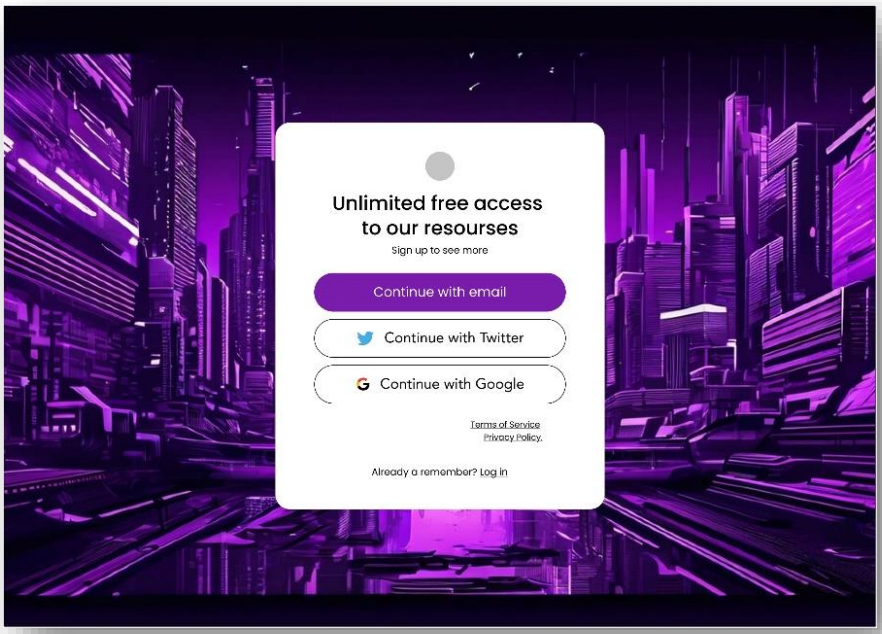


Figure 29 Automated Login Screen

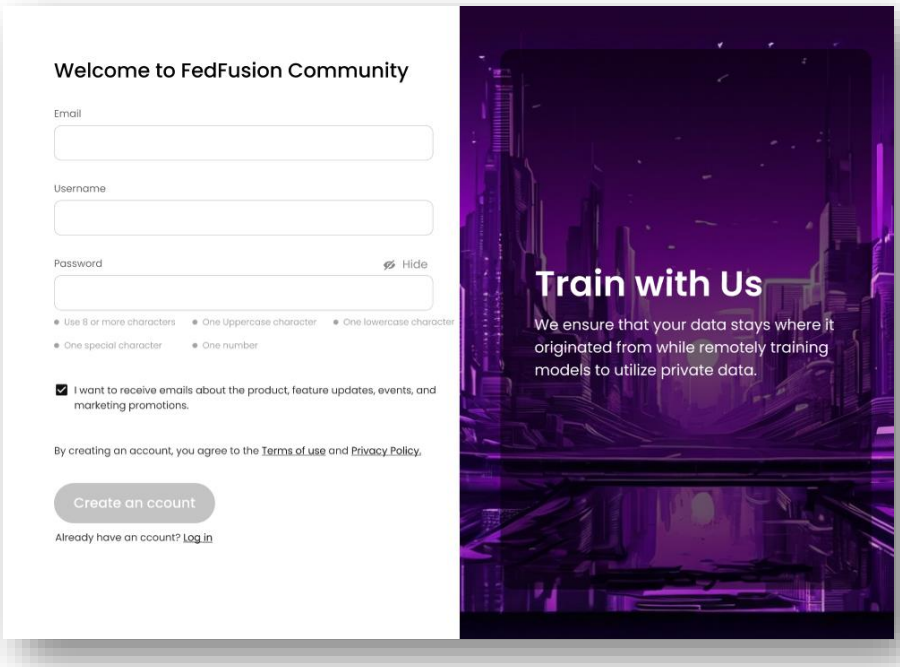


Figure 30 First-Time Login Screen

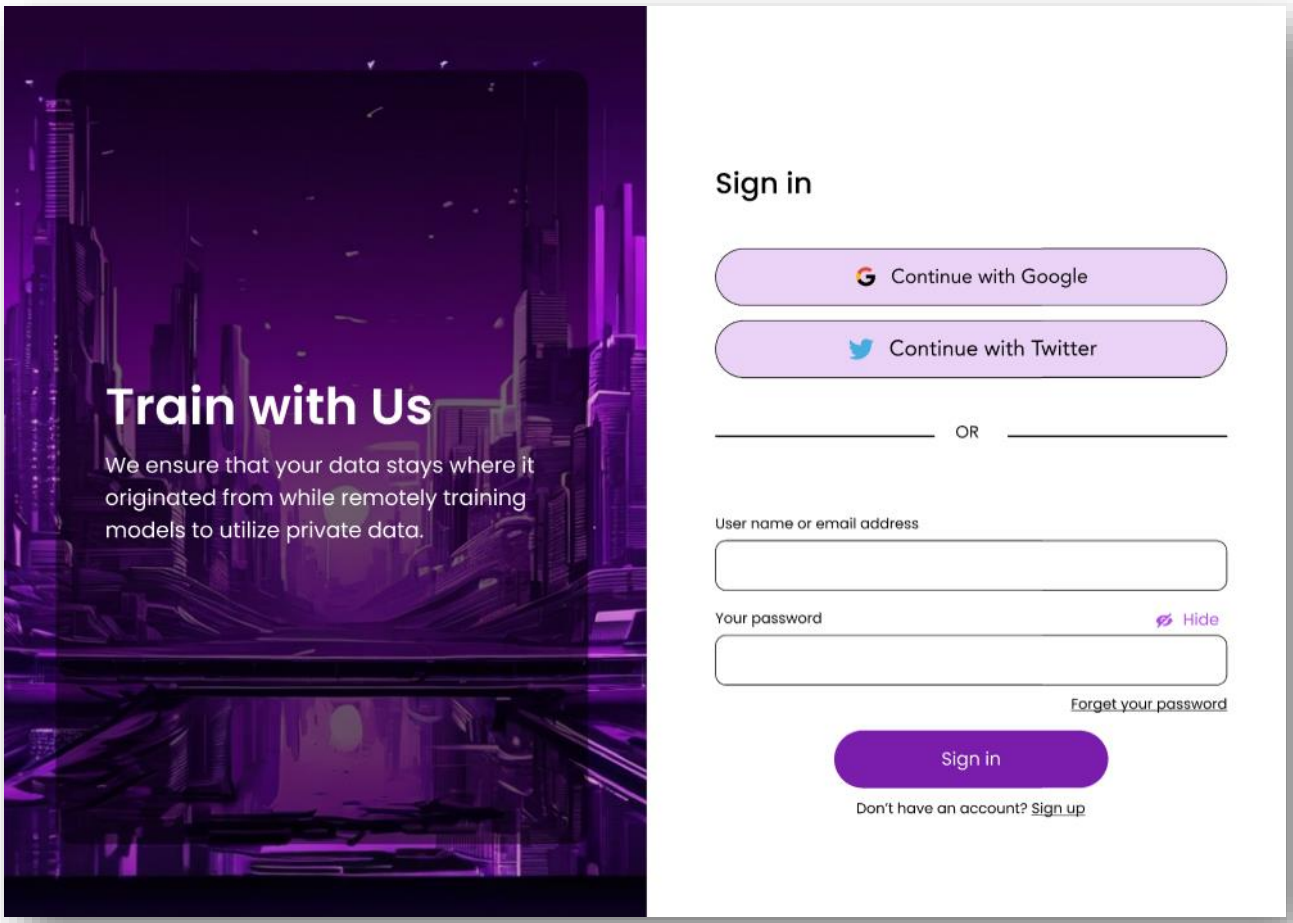


Figure 31 Sign-In Screen

HOME PAGE

Purpose: Introduce federated learning and encourage users to start a collaboration.
Key Objects:

- Header: "YOUR PRIVATE TRAINING PLATFORM".
- Overview of federated learning benefits.
- CTA Button: "START NOW" to proceed to **Role Selection**.

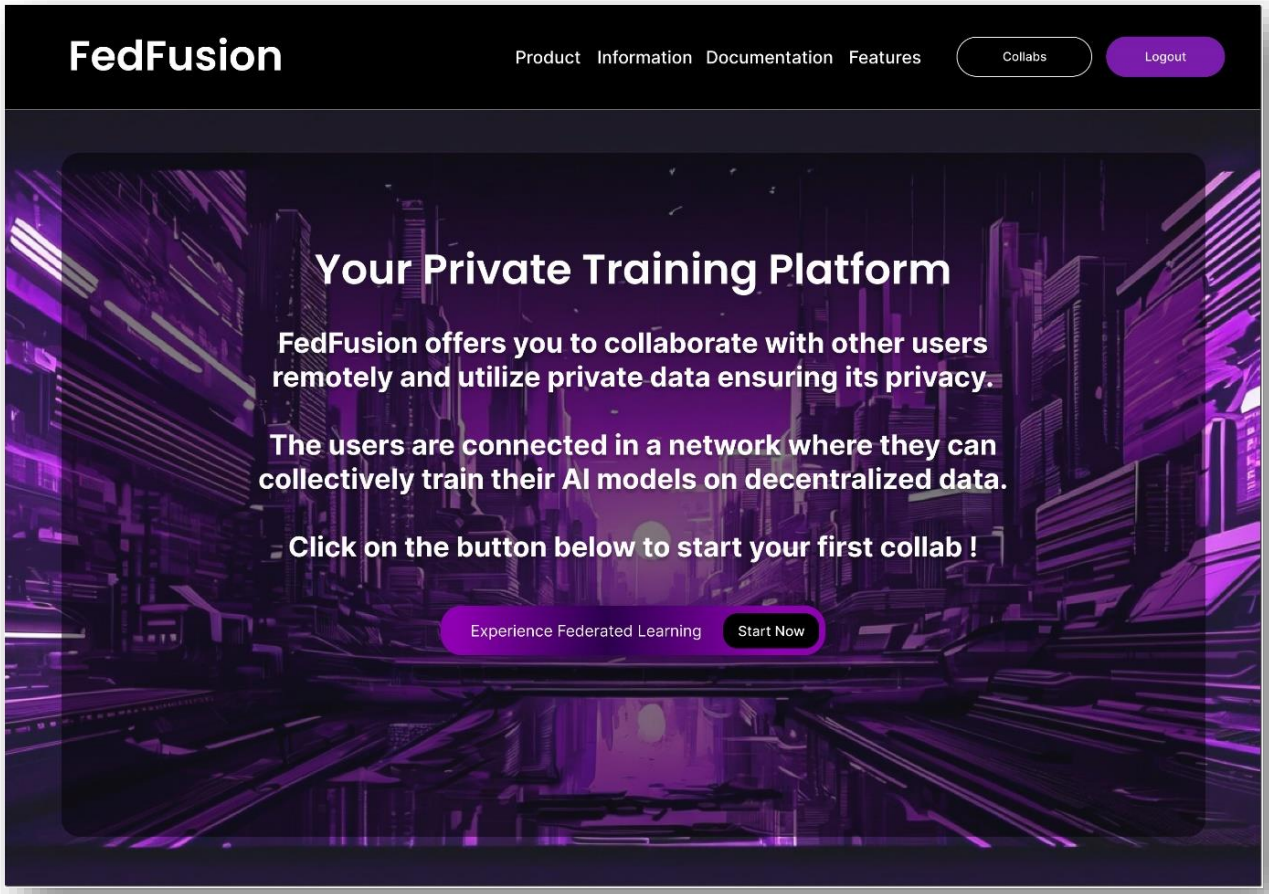


Figure 32 Home Page

CHOOSE ROLE SCREEN

- **Purpose**: Allow users to select their collaboration role.
- **Objects**:
- "Client" Button: Contribute data for training.
- "Server" Button: Host and manage the collaboration process.
- **System Feedback**:
- Selection confirmation with navigation to relevant screens.

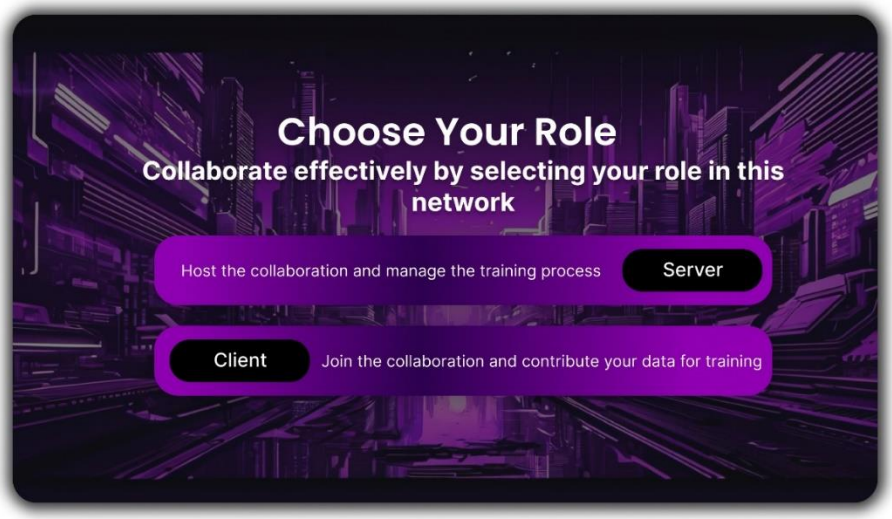


Figure 33 Initialization Screen

COLLABORATION DETAILS SCREEN

Server View:
Input Fields:

- BATCH SIZE, EPOCHS, INPUT COLUMN, OUTPUT COLUMN, NUMBER OF CLASSES.
- Validation: "Batch size must be greater or equal to 32."

Dropdowns: MODEL TYPE (e.g., Text/Image Classification).
Start Button: Initiates the server-side collaboration process.

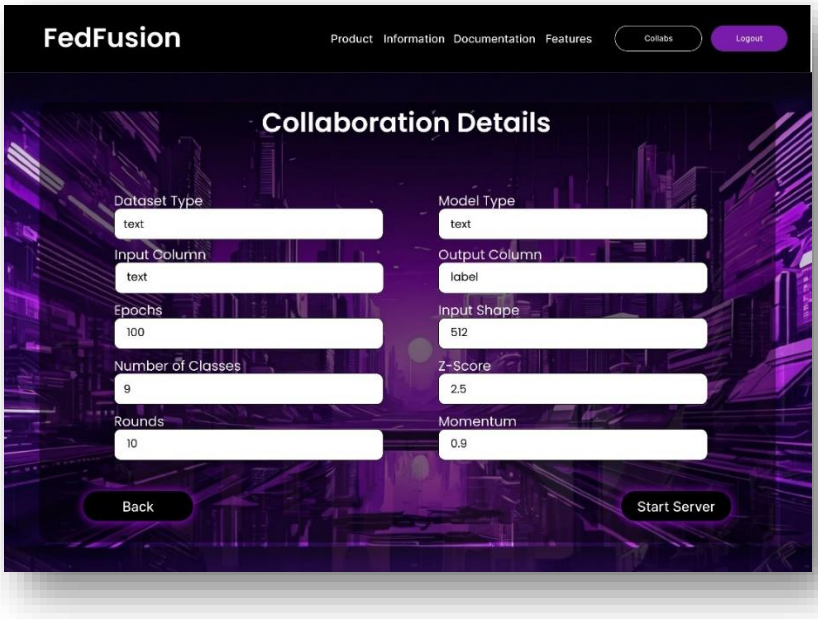


Figure 34 Server Setup Configuration Screen

Client View:
Input Fields:

- SERVER ADDRESS, DATASET PATH, INPUT/OUTPUT COLUMNS.
- Start Button: Connects the client to the server for data contribution.

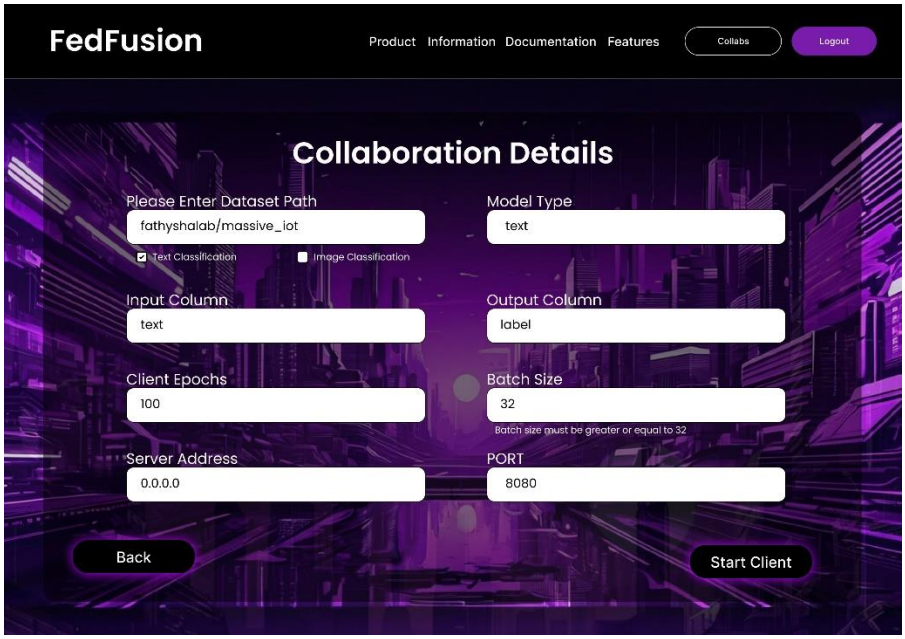


Figure 35 Client Setup Configuration Screen

ERROR HANDLING:

- "Fill out the required fields" if any input is missing.

Figure 36 Error Handling Notification

DATA PRIVACY OPTIONS SCREEN

Purpose: Allow users to choose between security modes.
Objects:

- "Open Mode" Option: Prioritizes speed without encryption.
- "Encryption Mode" Option: Ensures data privacy with RSA/AES encryption.

User Actions:

- Selecting “Encryption” enables the **Key Management Screen**.

Figure 37 Encryption Option Screen

KEY MANAGEMENT SCREEN

Purpose: Enable secure encryption through RSA and AES keys.
Objects:

- File Upload Fields:
- Upload Private RSA Key
- Upload Encrypted AES Key.

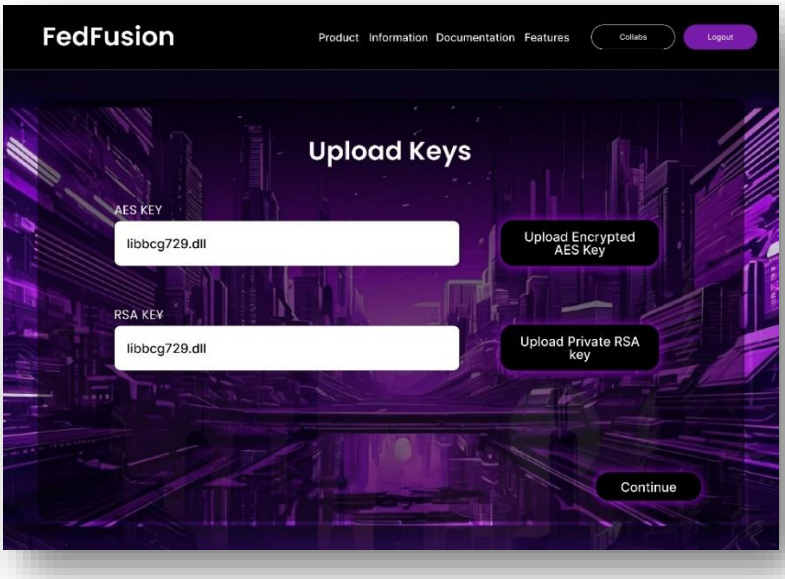


Figure 38 Key Details Screen

Download Buttons:

- "Download AES Key"
- "Download RSA Key"

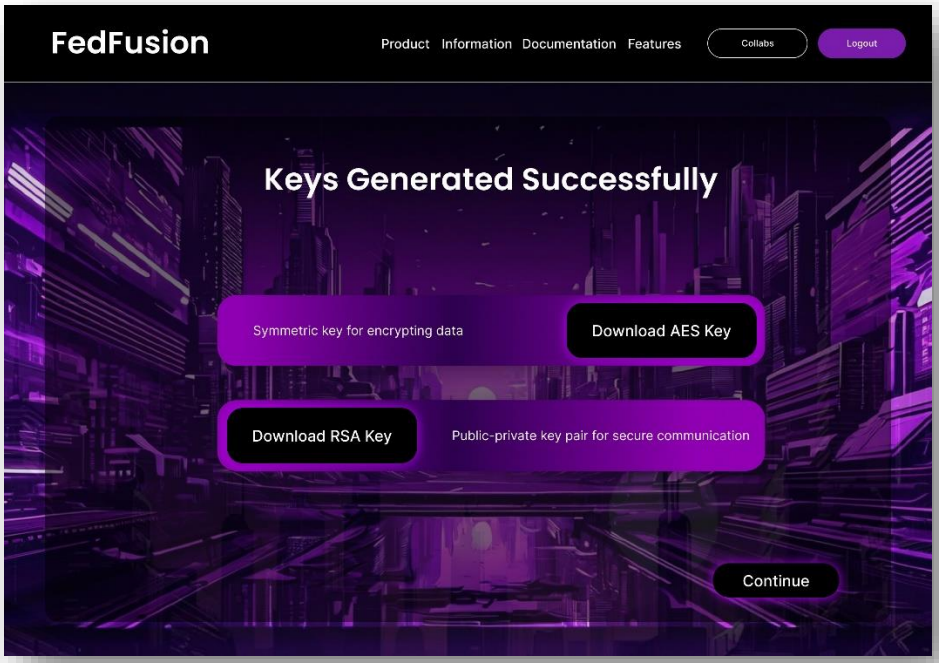


Figure 39 Key Storage Screen

Feedback:

- "Keys Uploaded Successfully."
- "Keys Generated Successfully."

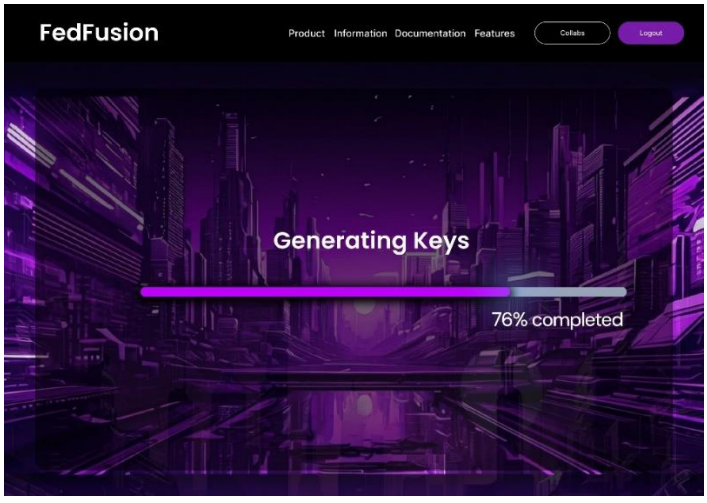


Figure 40 Key Generation Progress Screen

TRAINING LOGS AND VISUALIZATION SCREEN

Purpose: Provide real-time updates during the model training process.
Objects:

- Logs Display: Training progress per epoch.

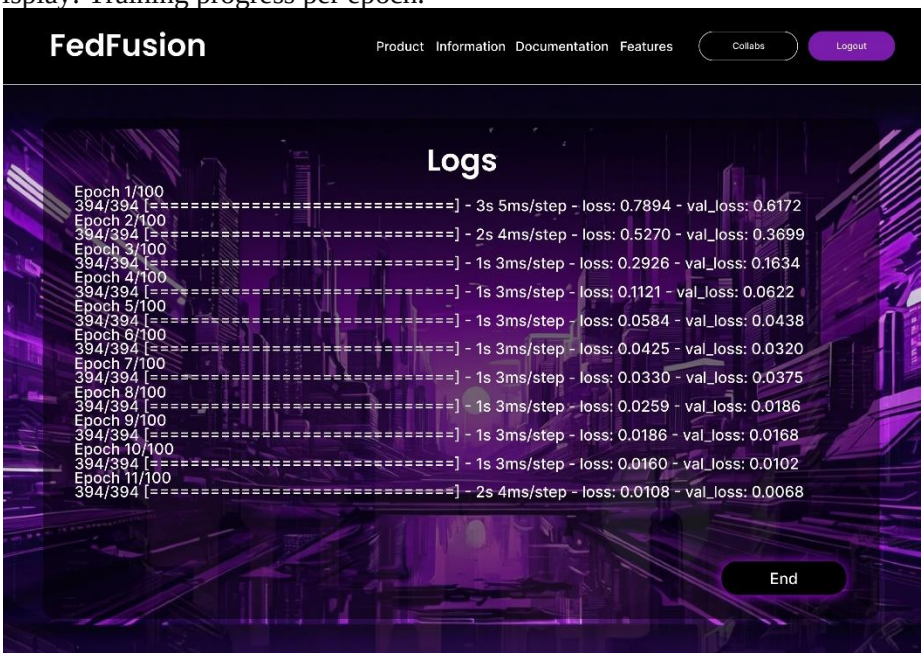


Figure 41 Logs Screen

Visualization: Live graphs of loss, accuracy, and other performance metrics.

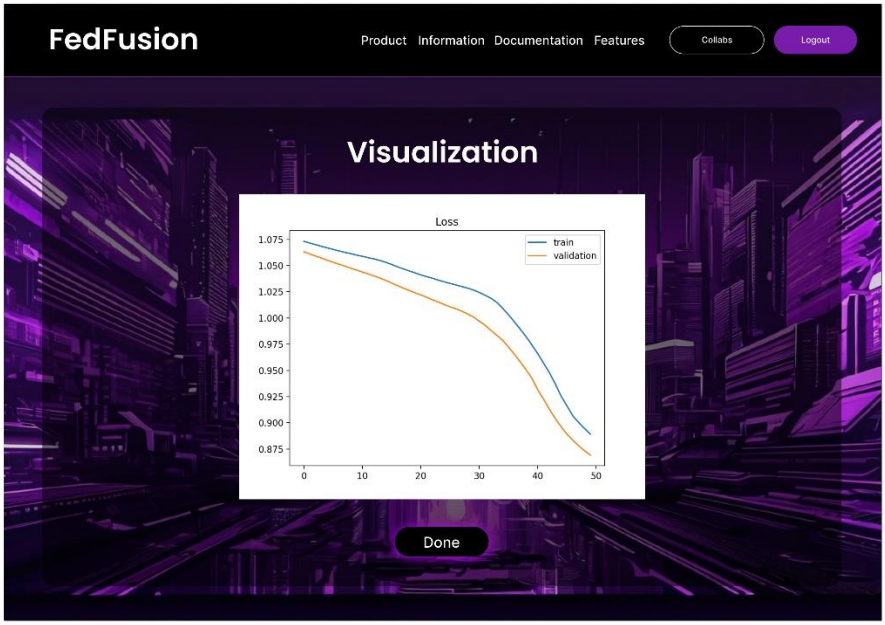


Figure 42 Metric Curves Screen

- Feedback: “Model Saved Successfully” at the end of training.

Figure 43 Model Save Success Screen

FEATURE INFORMATION SCREENS

Purpose: Educate users about platform features.
Key Features:

- Hugging Face Integration
- Real-Time Visualization
- Encrypted Parameter Passing
- Dynamic Node Collaboration

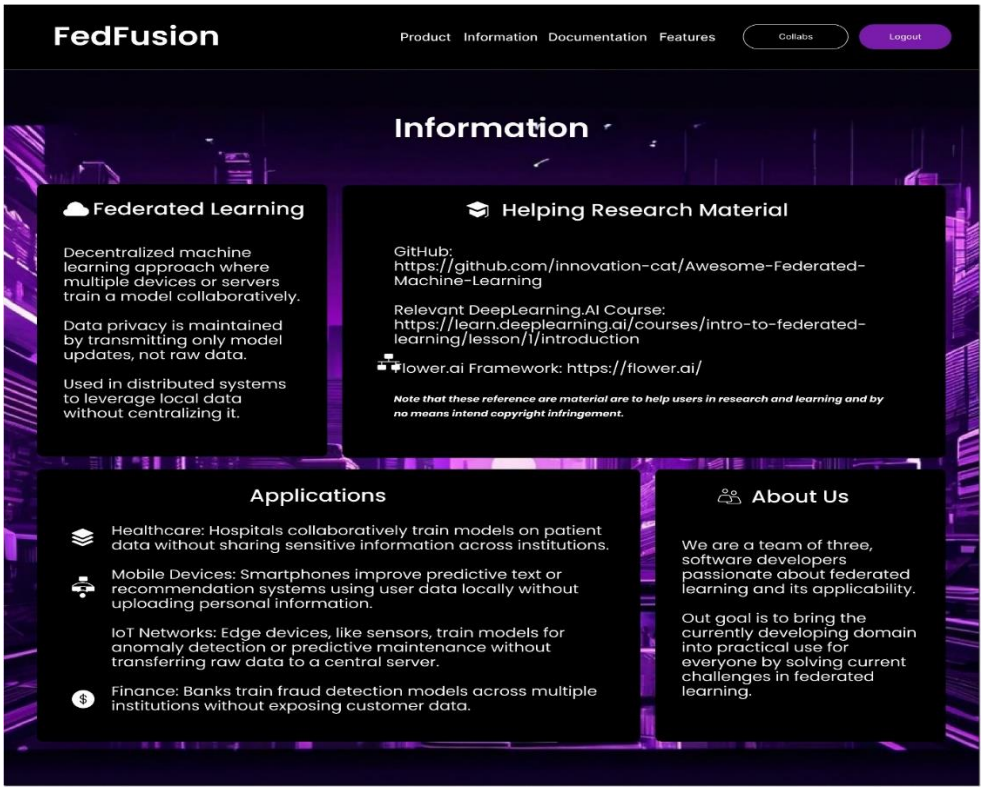


Figure 44 Information Page

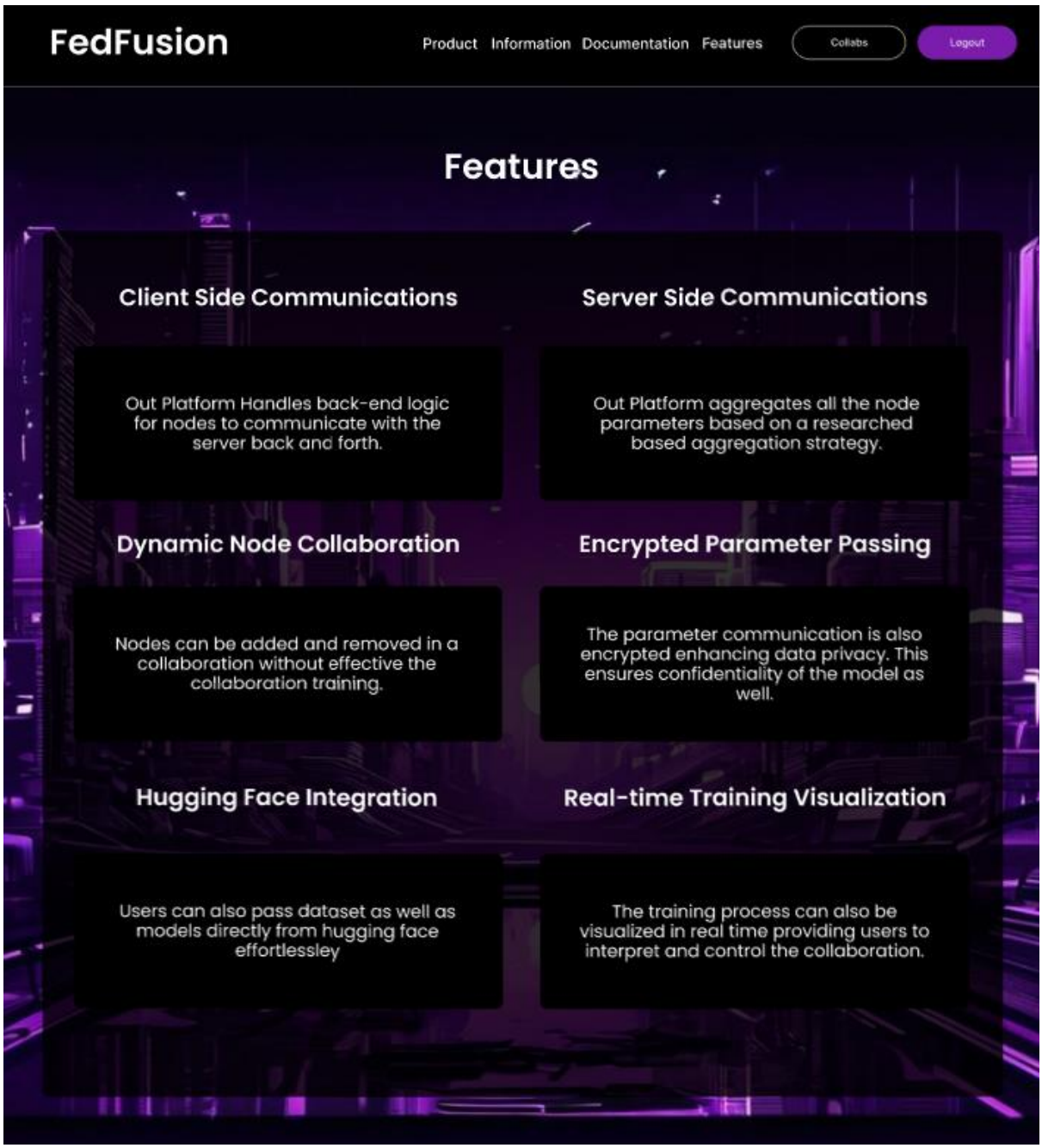


Figure 45 Functionality Feature

SCREEN OBJECTS AND ACTIONS

Screen	Object	Action	Feedback/Response
Login Screen	"Sign In" Button	Click to log in	Redirect to Home Page on success.
	"Forget Password" Link	Click to reset password	Navigate to Password Reset screen.
	Input Fields	Enter credentials	Error: "Invalid credentials."
	Social Media Buttons	Authenticate via Google/Twitter	Redirect to oAuth flow.
Home Page	"Start Now" Button	Begin collaboration	Navigate to Role Selection screen.
Choose Role Screen	"Client" Button	Select client role	Redirect to Client Collaboration screen.
	"Server" Button	Select server role	Redirect to Server Collaboration screen.
Collaboration Details	Input Fields (e.g., Batch Size)	Enter parameters	Error: "Batch size must be >= 32."
	"Start Server" Button	Initiate training	Show real-time logs.
	"Start Client" Button	Join server collaboration	Show real-time logs.
Data Privacy Options	"Open Mode"	Select fast training	Display mode confirmation.
	"Encryption"	Enable data security	Navigate to Key Management screen.
Key Management Screen	File Upload Buttons	Upload RSA and AES keys	Success: "Keys Uploaded Successfully."
	"Generate Keys" Button	Generate encryption keys	"Keys Generated Successfully."
Training Screen	Logs Display	View training progress	Dynamic updates: loss, accuracy per epoch.
	Visualization Graphs	Monitor training visually	Real-time performance graphs.
	"Model Saved Successfully"	Training completion feedback	Confirmation popup.

Table 3 Screen Objects and Action

NAVIGATION FLOW

The navigation flow for FedFusion is designed to guide the user seamlessly from login to collaboration setup and real-time training monitoring. Below, we detail the step-by-step journey from the user's perspective, covering each screen, actions, and feedback mechanisms.

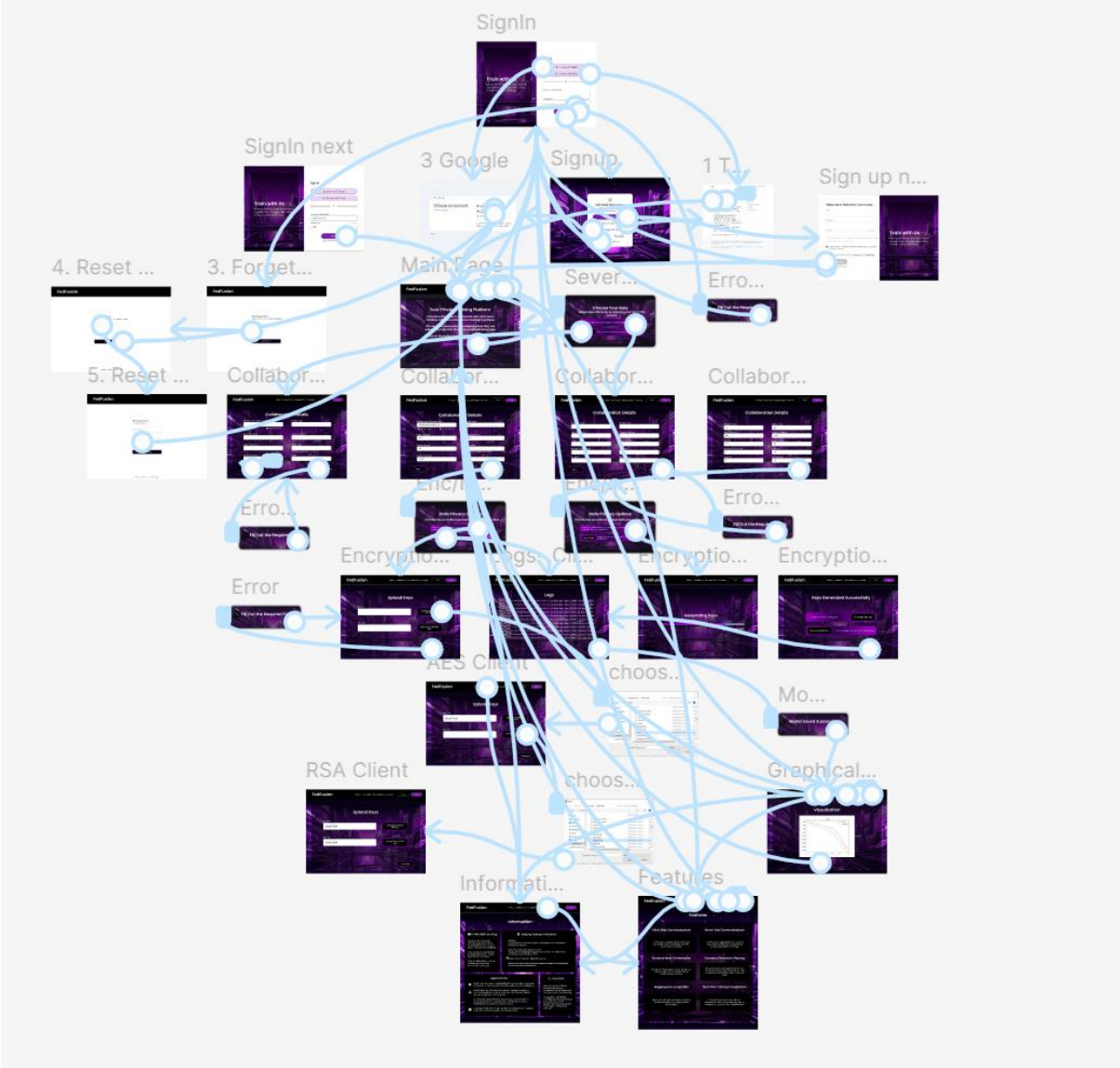


Figure 46 Navigation Flow

LOGIN AND SIGN-UP PROCESS

The user begins their journey with the **Login or Sign-Up page**, which supports multiple authentication methods to ensure accessibility.

Login Screen

User View:

- Upon opening the application, the user is presented with a login screen featuring:
- USERNAME/EMAIL input field.
- PASSWORD input field with an option to toggle visibility (Hide/Show).
- Social media login buttons (GOOGLE, TWITTER).
- FORGET PASSWORD? Link for recovery.

Action: The user can choose one of the following paths:

Enter Email and Password

- If the credentials are invalid, the system displays:
- ERROR MESSAGE: "Invalid credentials. Please try again."
- Once valid credentials are entered, the user is redirected to the **Home Page**.

Login via Google/Twitter

- Clicking a social media button initiates the oAuth process.
- After successful authentication, the user is redirected to the **Home Page**.

Sign-Up Screen

User View: If the user opts to create an account, they are directed to the Sign-Up page.

- Fields for EMAIL, USERNAME, and PASSWORD.
- Password requirements are displayed for guidance:
- At least 8 characters.
- Include one uppercase letter, one lowercase letter, one number, and one special character.
- Checkbox to opt-in for email notifications about product updates.
- "By creating an account, you agree to our Terms and Privacy Policy."

Action:

- The user fills in the form. If the password doesn't meet the criteria, an error message appears:
- ERROR MESSAGE: "Password must meet all requirements."
- After successful sign-up, the user is redirected to the **Login Screen** with a confirmation.

Password Recovery

User View: The user clicks FORGET PASSWORD? And is redirected to the Password Recovery page.

- Field to enter the registered email address.

Action: The user enters their email and clicks RESET PASSWORD.

- System Feedback: "Recovery email sent successfully."
- The user receives a link via email to set a new password, leading to the reset page.

HOME PAGE

After logging in, the user lands on the **Home Page**, which serves as the central hub.

User View:

- A welcoming message: "YOUR PRIVATE TRAINING PLATFORM".
- An overview of federated learning benefits.
- Call-to-action button: "START NOW".

Action:

- Clicking "START NOW" redirects the user to the **Choose Role Screen**.

ROLE SELECTION (CHOOSE ROLE SCREEN)

The **Choose Role Screen** allows users to define their participation in the collaboration.

User View:

- Two large buttons labeled:
- CLIENT: To contribute data for training.
- SERVER: To host and manage the collaboration.

Action:

- Clicking a role takes the user to the respective **Collaboration Details Screen**.

COLLABORATION DETAILS

Server Collaboration

User View: The user is presented with input fields to define collaboration parameters, such as:

- BATCH SIZE (with validation for minimum size of 32).
- NUMBER OF EPOCHS.
- INPUT AND OUTPUT COLUMNS.
- MODEL TYPE: Options for Text or Image Classification.

Actions:

- The user enters details. If a required field is missing, the system displays:
- ERROR MESSAGE: “Fill out the required fields.”
- Clicking START SERVER initiates the training process and redirects to the **Training Logs Screen**.

Client Collaboration

User View: The client enters the:

- SERVER ADDRESS.
- DATASET PATH.

Actions:

- Clicking START CLIENT initiates the training process and redirects to the **Training Logs Screen**.

DATA PRIVACY OPTIONS

Before starting the collaboration, the user is prompted to choose a data privacy mode.

User View:

- Two options:
- OPEN MODE: Training prioritizes speed without encryption.
- ENCRYPTION: Secures data with RSA and AES keys.

Action:

- Selecting ENCRYPTION redirects the user to the **Key Management Screen**.

KEY MANAGEMENT

User View: The interface provides options to upload or generate encryption keys.
Fields for clients:

- UPLOAD PRIVATE RSA KEY.
- UPLOAD AES KEY.
- Buttons for server:
- GENERATE KEYS (if no keys are uploaded).
- DOWNLOAD RSA KEY and AES KEY.

Actions:

- Upload keys or generate new ones.
- Successful upload generates the message:
- “KEYS UPLOADED SUCCESSFULLY.”

TRAINING LOGS AND VISUALIZATION

User View: The training screen provides:

- Real-time logs of epoch progress, including:
- LOSS and VALIDATION LOSS values.
- Dynamic graphs visualizing loss and accuracy metrics.
- Final message upon completion:
- "MODEL SAVED SUCCESSFULLY."

NAVBAR

The **Navbar** is a persistent navigation element available across most screens, providing users with quick access to key pages and functionalities.

STRUCTURE AND OPTIONS

DOCUMENTATION

Purpose: Redirect users to an external resource that provides a comprehensive explanation of federated learning.
Action:

- When clicked, it opens the Flower Labs tutorial on federated learning:
- What is Federated Learning? (Flower AI).

User Feedback: Opens in a new browser tab, ensuring the current session on FedFusion remains active.

COLLABS

Purpose: Redirect users to the *Home Page*, where they can restart the collaboration process.
Action:

- Clicking the *Collabs* button takes users back to the landing page with the "Start Now" button and federated learning overview.

User Feedback: No data loss occurs; the user can restart their workflow seamlessly.

PRODUCTS

Purpose: Similar to the *Collabs* button, this redirects to the *Home Page*.
Action:

- Clicking *Products* navigates the user to the landing page.

User Feedback: Offers continuity in navigation for users exploring platform features.

INFORMATION

Purpose: Provides users with access to the *Information Page*, detailing the platform's purpose, federated learning applications, and team background.
Action:

- Clicking the *Information* button opens a dedicated page with:
- Overview of federated learning concepts.
- Use cases (Healthcare, IOT, Mobile devices, Finance).
- Team details and mission statement.

User Feedback: Presents static content for user reference without redirection to external sites.

FEATURES

Purpose: Redirects to the *Features Page*, outlining key functionalities of the FedFusion platform.
Action:

- Clicking the *Features* button navigates to a page showcasing:
- Real-time training visualization.
- Hugging Face integration for datasets and models.
- Encrypted parameter passing for secure communication.
- Dynamic node collaboration.

User Feedback: Ensures users can access and understand the advanced functionalities of the platform.

USAGE FLOW OF NAVBAR

1. **Documentation**: A user interested in learning more about federated learning clicks the *Documentation* button, which opens the Flower AI tutorial in a new tab.
2. **Collabs and Products**: If the user wishes to return to the starting point (e.g., after exploring other pages), they click either *Collabs* or *Products*, which takes them back to the *Home Page*.
3. **Information**: For a quick overview of federated learning applications and the team behind the platform, the user clicks *Information*.
4. **Features**: To explore platform functionalities, the user clicks *Features*, where they can view detailed explanations of tools like encrypted training and Hugging Face integration.

ACTIVITIES AND USER TASKS

The key user activities and their steps are outlined below:

1. Sign Up/Login

- Enter credentials or sign in via social media.
- Recover password if required.

2. Role Selection

- Choose to act as a client or Server.

3. Collaboration Setup

- **Server**: Input batch size, epochs, model type, and encryption options.
- **Client**: Input server address and dataset details.

4. Secure Collaboration

- Upload RSA/AES keys for encryption.
- Generate and download encryption keys.

5. Monitor Training

- View live logs and graphs for model progress.
- Receive feedback upon training completion.

FEEDBACK AND VALIDATION SUMMARY

Scenario	Feedback/Error Message
Missing Fields	“Fill out the required fields.”
Incorrect Login Credentials	“Invalid credentials.”
Batch Size Too Small	“Batch size must be >= 32.”

Key Upload Success	“Keys Uploaded Successfully.”
Training Completion	“Model Saved Successfully.”

Table 4 Feedback Messages

The FedFusion UI provides an intuitive and efficient way for users to engage in federated learning. With a focus on ease of use, error validation, and real-time feedback, the platform ensures a seamless collaboration experience for both servers and clients.

CROSS-REFERENCE OF COMPONENTS TO SRS REQUIREMENTS

Requirement Code	Functional Requirement Description	System Component	Data Structures
FR-01	Support secure communication between clients and server	Crypto Module	AES Key, RSA Keys
FR-02	Enable decentralized training for multiple clients	Client Module, Server Module	Client Object, Global Model
FR-03	Provide anomaly detection during model updates	CustomFedAvg Class	Metrics (Accuracy, Loss)
FR-04	Facilitate dataset preprocessing and splitting	Data Module	Dataset Object
FR-05	Validate encryption and decryption processes	Testing Module	Encrypted Weights, Decrypted Weights
FR-06	Allow modular addition of attack simulations	Attack Module	Simulated Gradients, Poisoned Data
FR-07	Display real-time training metrics	UI Module	Metrics (Loss, Accuracy)
FR-08	Maintain a persistent log of model configurations and results	Logging Module	Log Object, Configuration JSON

FR-01: SUPPORT SECURE COMMUNICATION BETWEEN CLIENTS AND SERVER

- **Status**: **InProgress**
- **Reason**: The provided code does include implementation related to secure communication protocols or cryptographic modules.

FR-02: ENABLE DECENTRALIZED TRAINING FOR MULTIPLE CLIENTS

- **Status**: **Done**
- **Reason**: The code includes classes and data structures (e.g., **client message**, **server message**, **fit ins**, **evaluate ins**) that facilitate communication between clients and the server, indicating support for decentralized training.

FR-03: PROVIDE ANOMALY DETECTION DURING MODEL UPDATES

- **Status**: **InProgress**
- **Reason**: There is no mention of anomaly detection mechanisms or classes related to monitoring model updates in the provided code.

FR-04: FACILITATE DATASET PREPROCESSING AND SPLITTING

- **Status**: **Done (Image Data Based)**

- **Reason**: The code does include data structures or classes for dataset preprocessing or splitting.

FR-05: VALIDATE ENCRYPTION AND DECRYPTION PROCESSES

- **Status**: **Done**
- **Reason**: There is no implementation for validating encryption and decryption processes in the provided code.

FR-06: ALLOW MODULAR ADDITION OF ATTACK SIMULATIONS

- **Status**: **InProgress**
- **Reason**: The code does include any modules or classes for simulating attacks or handling poisoned data.

FR-07: DISPLAY REAL-TIME TRAINING METRICS

- **Status**: **Done**
- **Reason**: The **fit res** and **evaluate res** classes include metrics (e.g., accuracy, loss) that can be used to display real-time training metrics.

FR-08: MAINTAIN A PERSISTENT LOG OF MODEL CONFIGURATIONS AND RESULTS

- **Status**: **Required**
- **Reason**: The code does not include any logging mechanisms or data structures for maintaining a persistent log of configurations and results.

RESULTS

Overall, the backend in completely functional with modules requiring testing. Front-end User Interface has also been developed. The majority of the functional working system has been implemented which is more then 50% of the project. We will further be generalizing the platform to cater more datasets and perform improved attack simulations. Moreover, we will be replicating baseline results of federated learning scenarios uncovered in paper reviews of security aspect of federated learning through our platform to compare with state-of-art research of Privacy in Artificial Intelligence and looking to contribute to improvement of those results if possible.

APPENDIX

APPENDIX A: PROJECT OVERVIEW

PROJECT TITLE

FedFusion: Secure Collaboration using Enhanced Federated Learning

AUTHORS

- Muhammad Saadan (368560)
- Misbah Juwayriyyah (382416)
- Wasif Mehmood (367315)

SUPERVISORS

- Mohsin Kamal
- Maajid Maqbool

INSTITUTION

National University of Sciences & Technology (NUST)
Bachelor of Engineering in Software Engineering (2024-2025)
Department of Computing, School of Electrical Engineering and Computer Science

APPENDIX B: FUNCTIONAL REQUIREMENTS SUMMARY

1. FR-01: Implement secure communication protocols using AES and RSA keys to protect data transmission between clients and the server. (Status: In Progress)

2. FR-02: Enable multiple clients to collaboratively train models on their local data while communicating with a central server. (Status: Done)

3. FR-03: Integrate anomaly detection mechanisms to monitor and identify irregularities during model updates. (Status: In Progress)

4. FR-04: Provide functionalities for preprocessing and splitting datasets to prepare them for training. (Status: Done)

5. FR-05: Validate the encryption and decryption processes to ensure the security of model parameters. (Status: Done)

6. FR-06: Allow for the modular addition of attack simulations to test the robustness of the federated learning system. (Status: In Progress)

7. FR-07: Display real-time training metrics, such as loss and accuracy, to monitor model performance during training. (Status: Done)

8. FR-08: Maintain a persistent log of model configurations and results for future reference and analysis. (Status: Required)

APPENDIX C: MODULES OVERVIEW

1. SERVER MODULE

- Manages the central server for federated learning.
- Coordinates training across multiple clients.
- Implements custom federated averaging strategy.

2. CLIENT MODULE

- Implements client-side logic for federated learning.
- Handles data preparation, model training, and communication with the server.
- Supports gradient leakage simulation.

3. DATA MODULE

- Provides utility functions for loading, preparing, and processing datasets.
- Integrates with Hugging Face's dataset library.

4. MODEL MODULE

- Contains functions to build and compile various model architectures using TensorFlow and Keras.

5. CRYPTO MODULE

- Handles encryption and decryption of model parameters and AES keys.
- Ensures secure communication between clients and the server.

6. ATTACK MODULE

- Simulates gradient leakage attacks and data poisoning.
- Tests the robustness of the federated learning system.

7. TESTING MODULE

- Provides functionality to test encryption and decryption processes.

APPENDIX D: DESIGN METHODOLOGY

DESIGN METHODOLOGY

- **Object-Oriented Programming (OOP)**: Encapsulation, reusability, abstraction, maintainability, and real-world modeling.

PROCESS MODEL

- **Agile Development**: Emphasizes iterative development, flexibility, collaboration, user-centric focus, and continuous integration/testing.

APPENDIX E: KEY FEATURES

SECURE COMMUNICATION

- Utilizes RSA and AES encryption for secure data transmission.

MULTI-CLIENT SUPPORT

- Allows multiple clients to train models on local data while collaborating with a central server.

ANOMALY DETECTION

- Implements methods for detecting anomalies in client updates using Z-score analysis.

REAL-TIME MONITORING

- Displays training metrics (loss, accuracy) during the training process.

CUSTOMIZATION AND EXTENSIBILITY

- Framework designed to be flexible and easily extendable for various use cases.

APPENDIX F: DATA STRUCTURES

MAJOR DATA ENTITIES

- **Client**: Represents individual clients with attributes like client_id, local_data, model_weights, and metrics.
- **Server**: Central server managing the global model and aggregating updates.
- **Model**: Defines machine learning models with attributes like input_shape and architecture.
- **Dataset**: Represents datasets used for training and testing.

- **Communication Parameters**: Includes attributes for secure communication, such as encrypted_weights and aes_key.

APPENDIX G: USER INTERFACE DESIGN

KEY SCREENS

- **Login and Registration**: Supports multiple authentication methods.
- **Home Page**: Introduces federated learning and encourages collaboration.
- **Role Selection**: Allows users to choose their role (Client or Server).
- **Collaboration Setup**: Input fields for defining collaboration parameters.
- **Training Logs and Visualization**: Provides real-time updates during model training.

APPENDIX H: FUTURE WORK

- Generalizing the platform to support more datasets.
- Improving attack simulations.
- Replicating baseline results of federated learning scenarios for comparison with state-of-the-art research.