Logbook – Privacy Preservation Module of R25 -039

Project ID: R25 - 039

Project Title: Data-Privacy Focused Federated Learning Framework for Industrial IoT

Student Details:	
Names:	Student IDs:
Mendis H.R.M	IT21822612

Supervisor: Mr. Amila Seneratha

Co-Supervisor: Mr. Tharaniyawarma Kumaralingam

Date of Submission: 2025

Contents

1. Group Details

Student Details:		
Names:	Student IDs:	Research Component
Nanayakkara Y.D.T. D	IT21826368	Attack Defense and Resillence
Mendis H.R.M	IT21822612	Privacy Preseravation
Weerasinghe K.M	IT21831904	Secure Aggrigation
Dissanayaka K.D.A.R. A	IT21828348	Secure Communicaiton and Protocol Enforcement

2. Project Details

Topic - Data-Privacy Focused Federated Learning Framework for Industrial IoT

Aim – To develop a product that going to full fill the research

Deliverables – Federated Learning Framework designed for industrial internet of things

This project was initiated to develop a secure and private **Federated Learning (FL) framework** specifically for **Industrial IoT (IIoT)** environments.

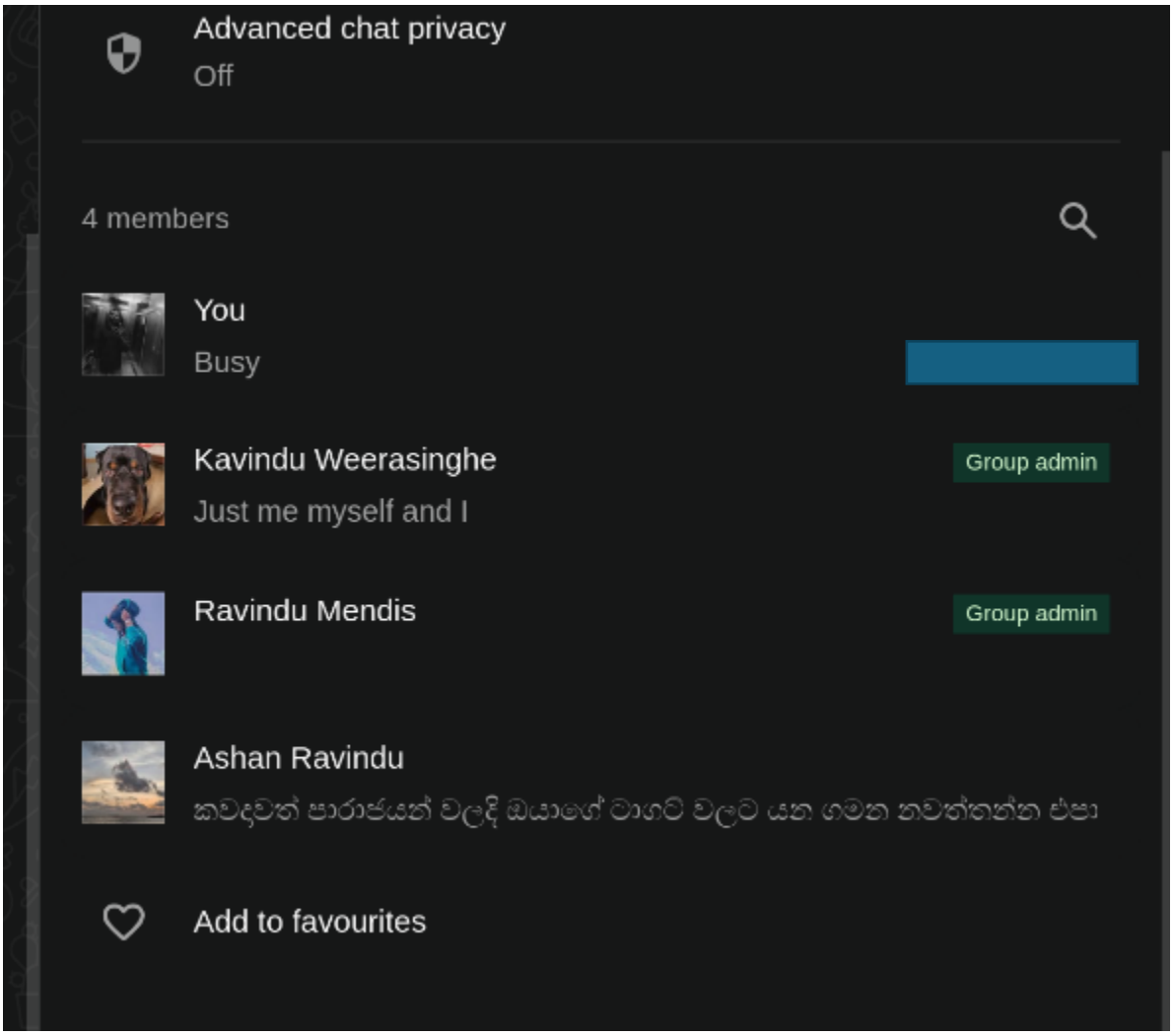
The Challenge: Traditional AI methods require centralizing sensitive factory data, which poses major **privacy risks** and clashes with the distributed nature of industrial operations. Existing FL solutions are insufficient because they fail to simultaneously provide robust security, data privacy, and efficient operation on **resource-limited IIoT devices**.

The Solution: The developed framework is a multi-layered system that provides **end-to-end protection**.

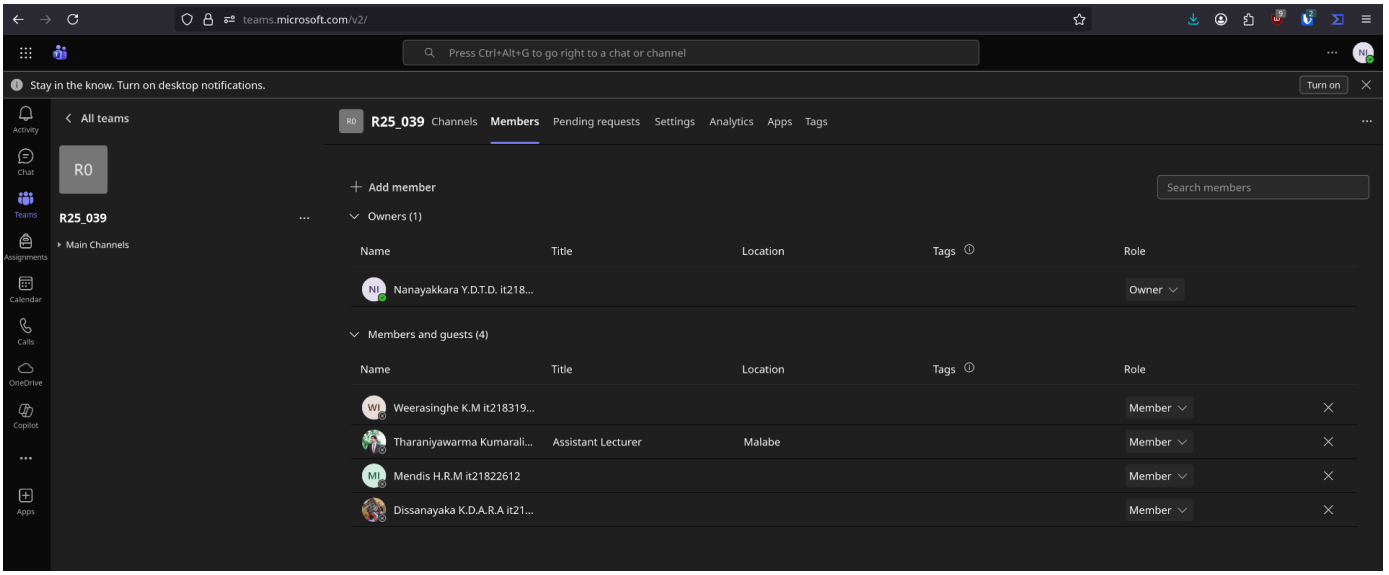
- It uses techniques like **Differential Privacy (DP)** and **Homomorphic Encryption (HE)** to guarantee data confidentiality.
- It implements a robust protocol that uses **client/server validation** to actively block cyber threats such as **Model Poisoning and Byzantine Attacks**.
- The system is optimized for **efficiency** to reduce overhead on IIoT devices.

3. Communication Methods

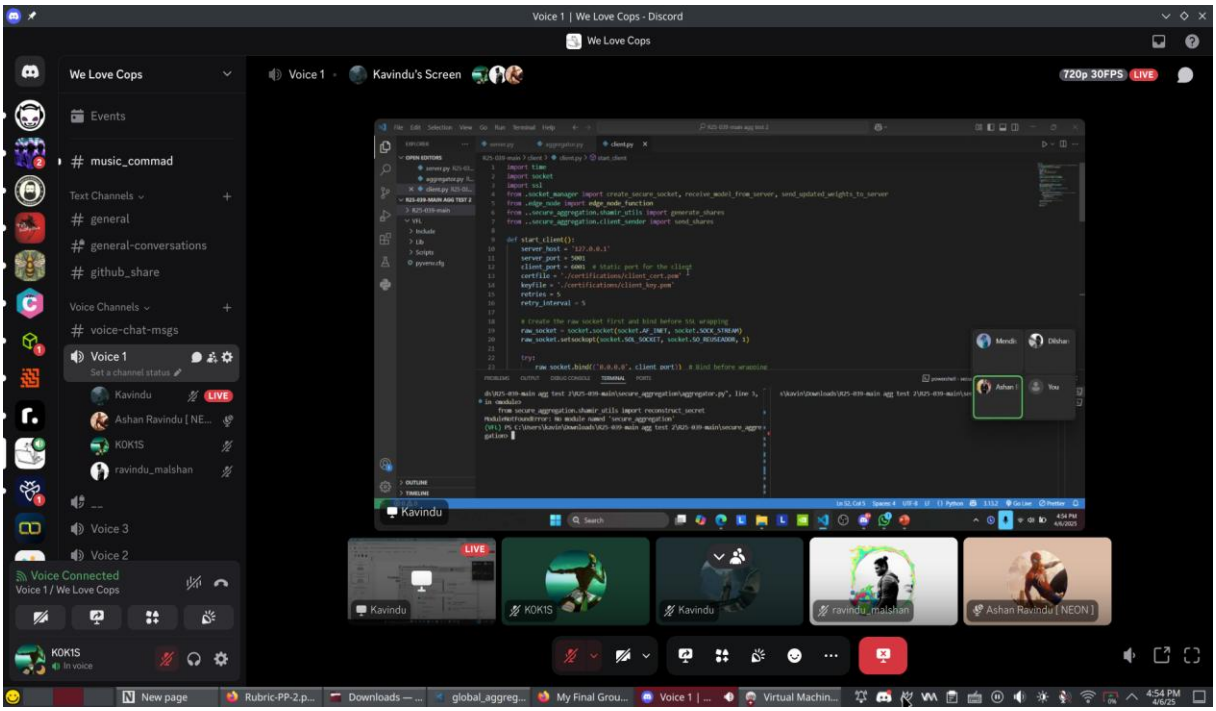
WhatsApp Group – Team



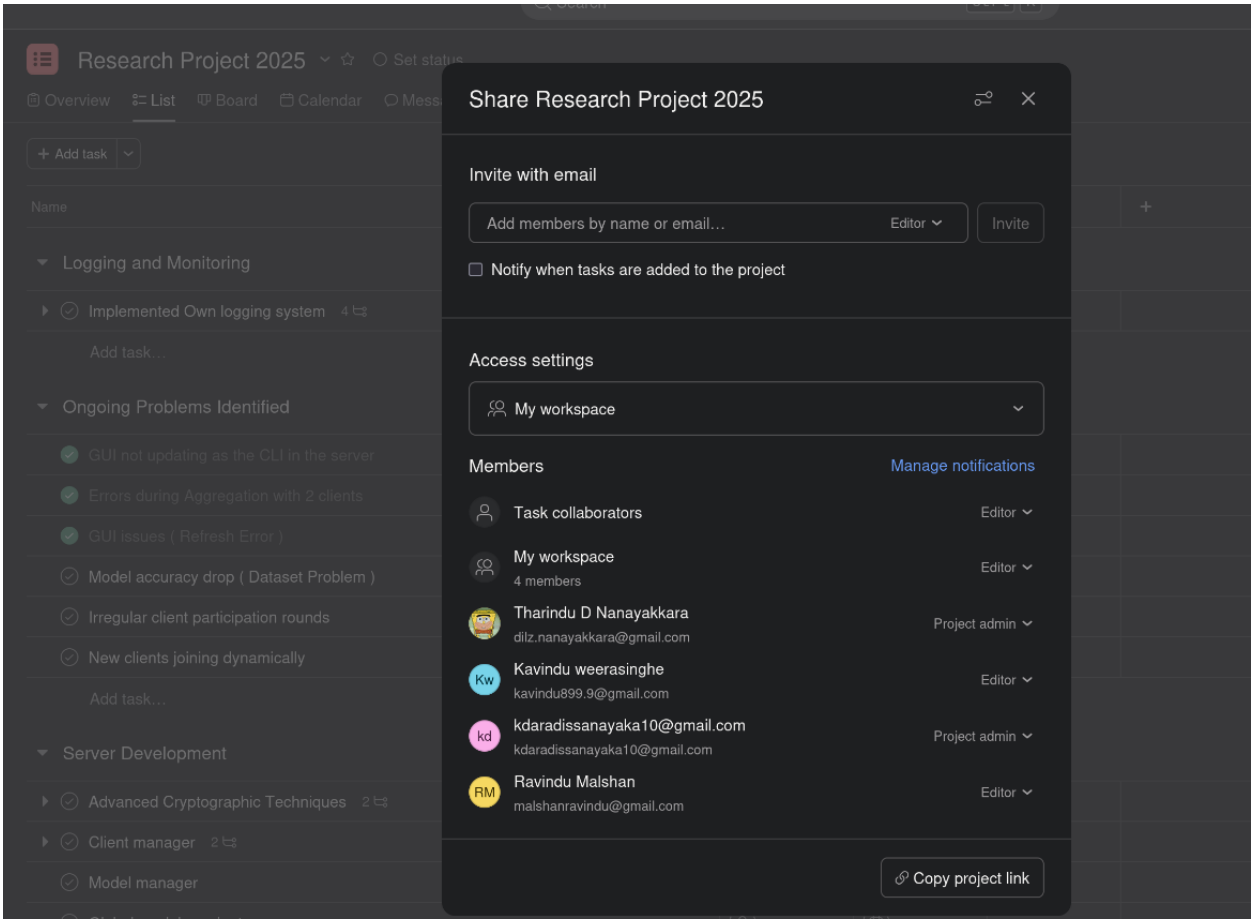
Microsoft Teams - All



Group Meetings – Discord – Team



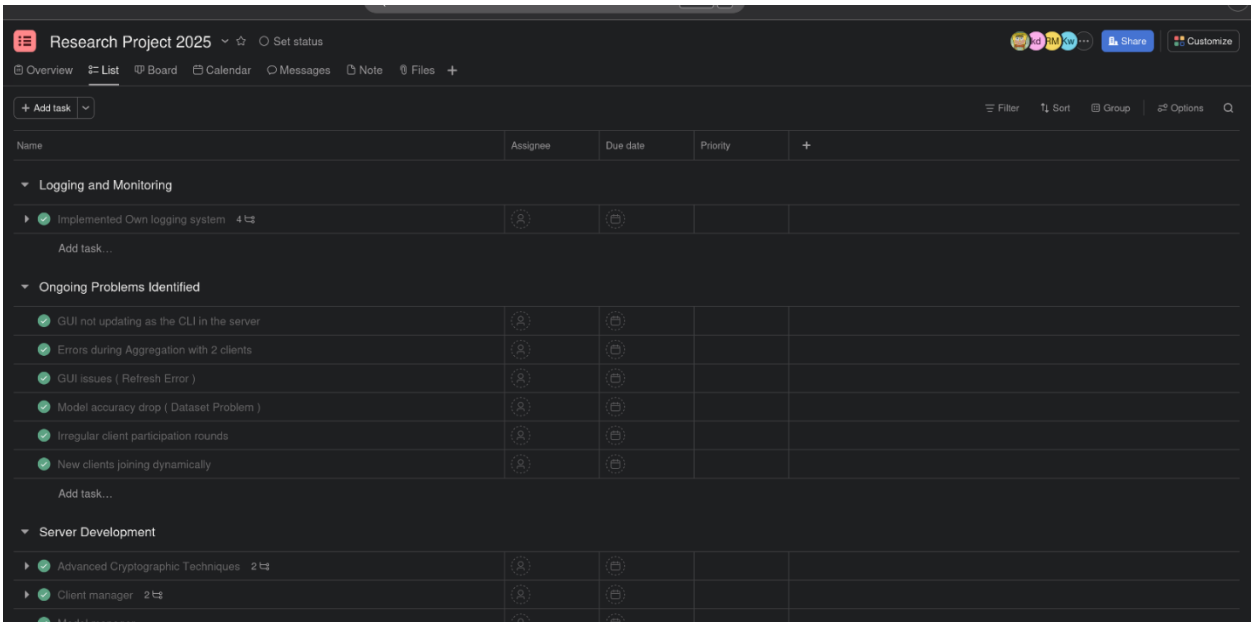
Asana – Task Assigning – Team



4. Meetings With Supervisors

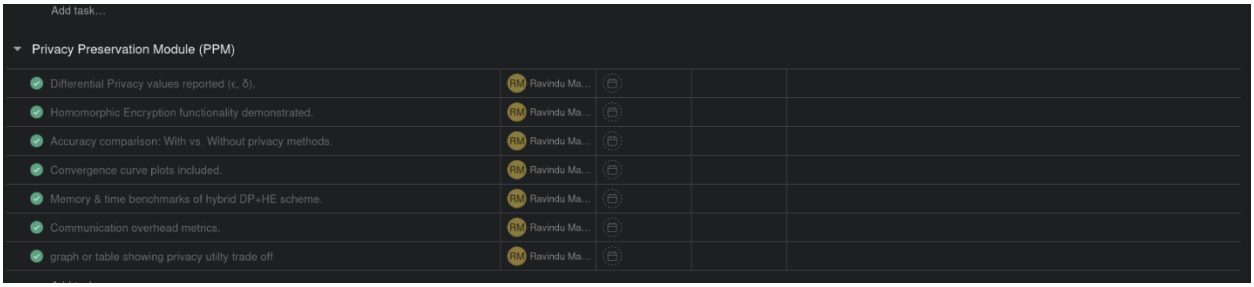
All the meetings were conducted in person and only WhatsApp calls were taken to organize the meeting

5. Task Details



Privacy Preservation Module was assgined.

5.1 Personal task Assigning and Completion

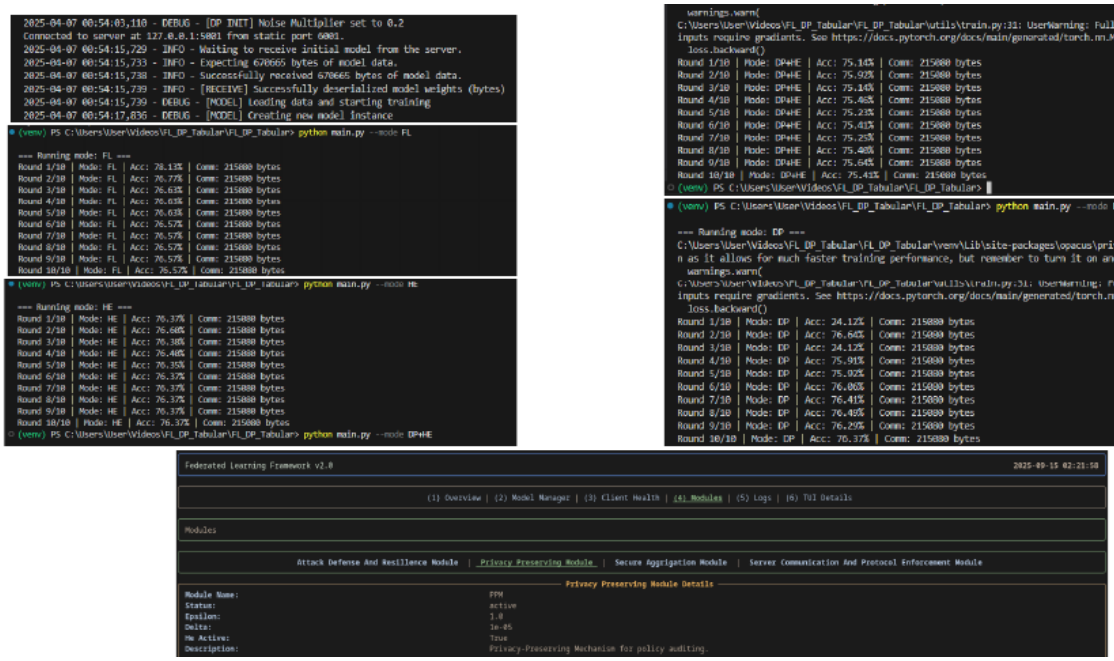


6. System Details

6.1 System completion status

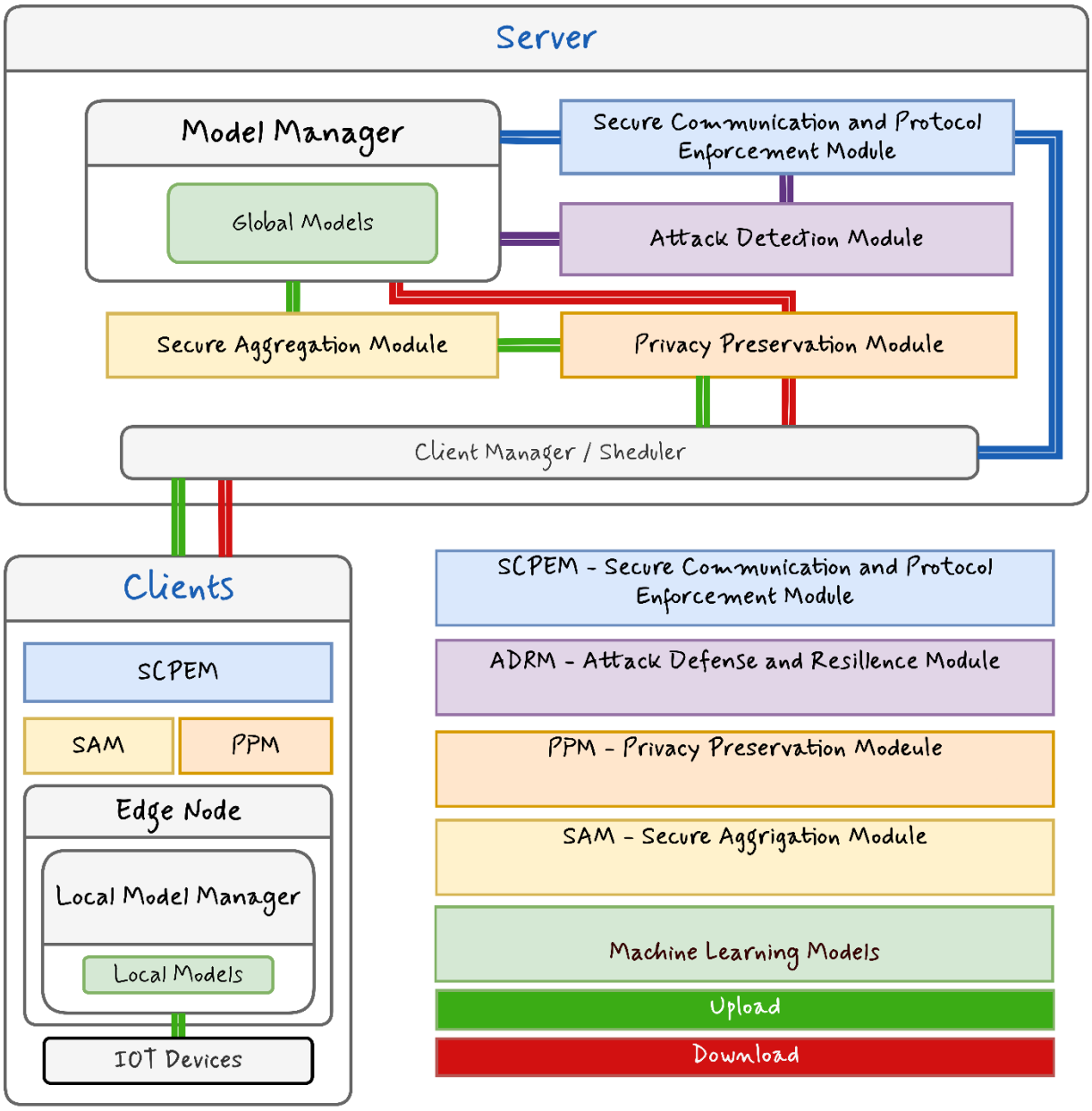
Finished

PPM TUI

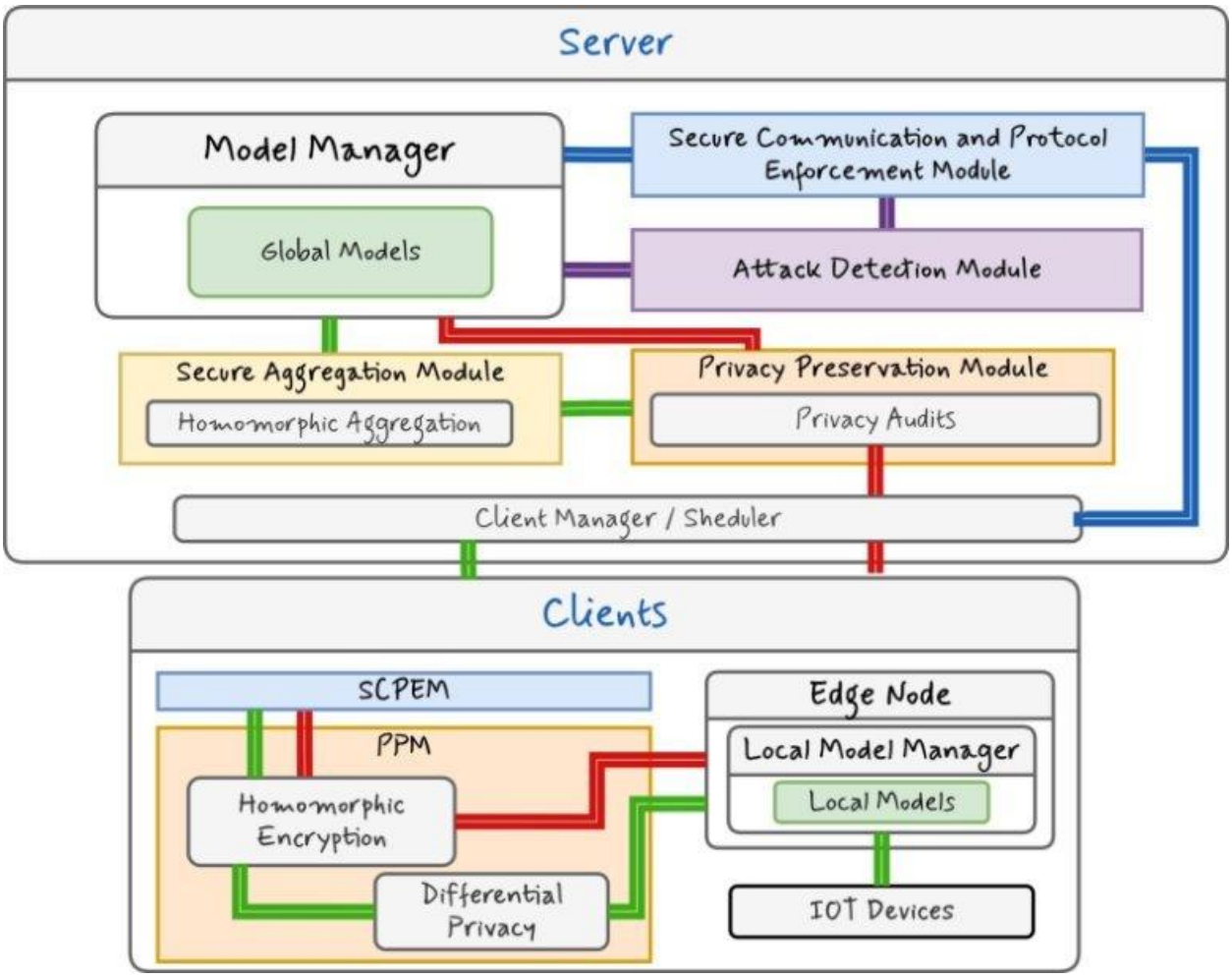


6.2 System Design

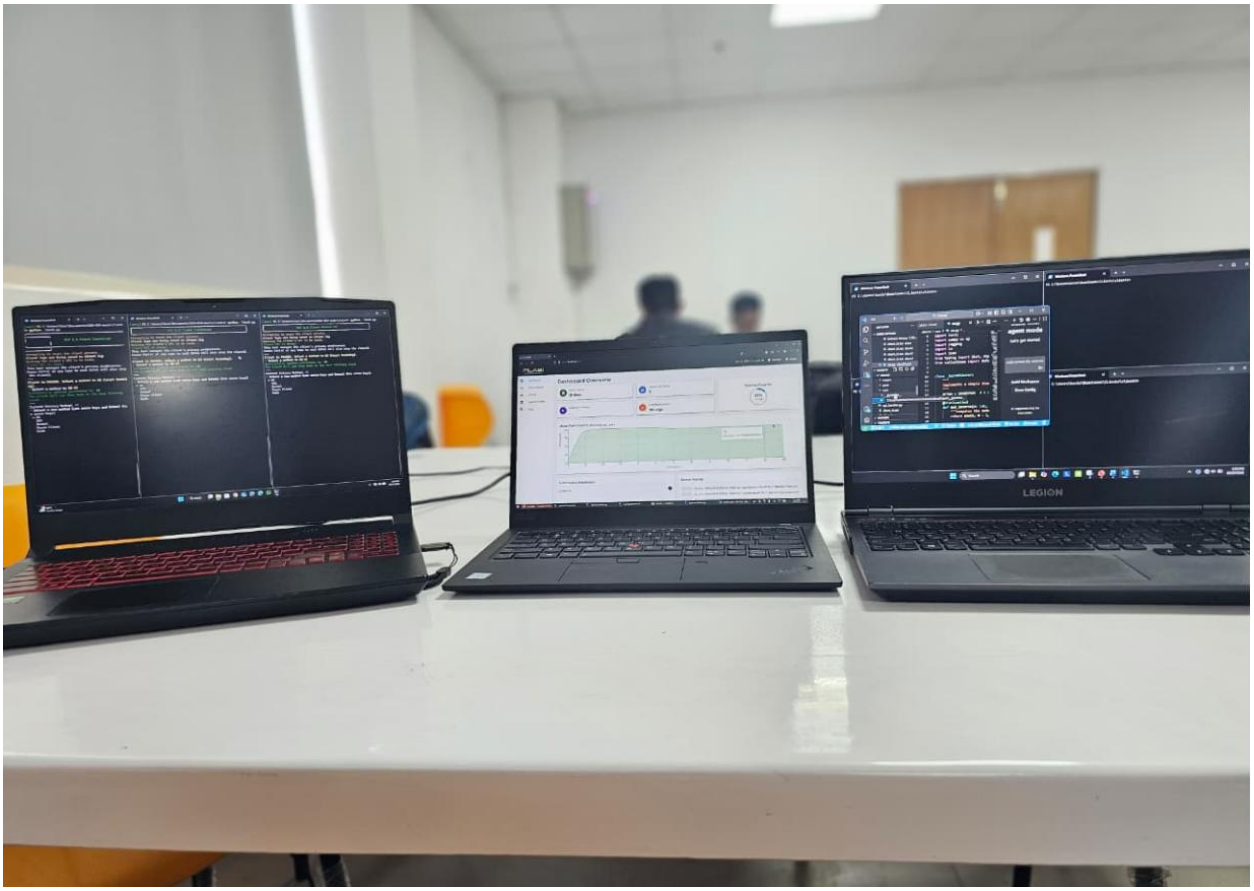
I. System Architecture



II. Module Architecture

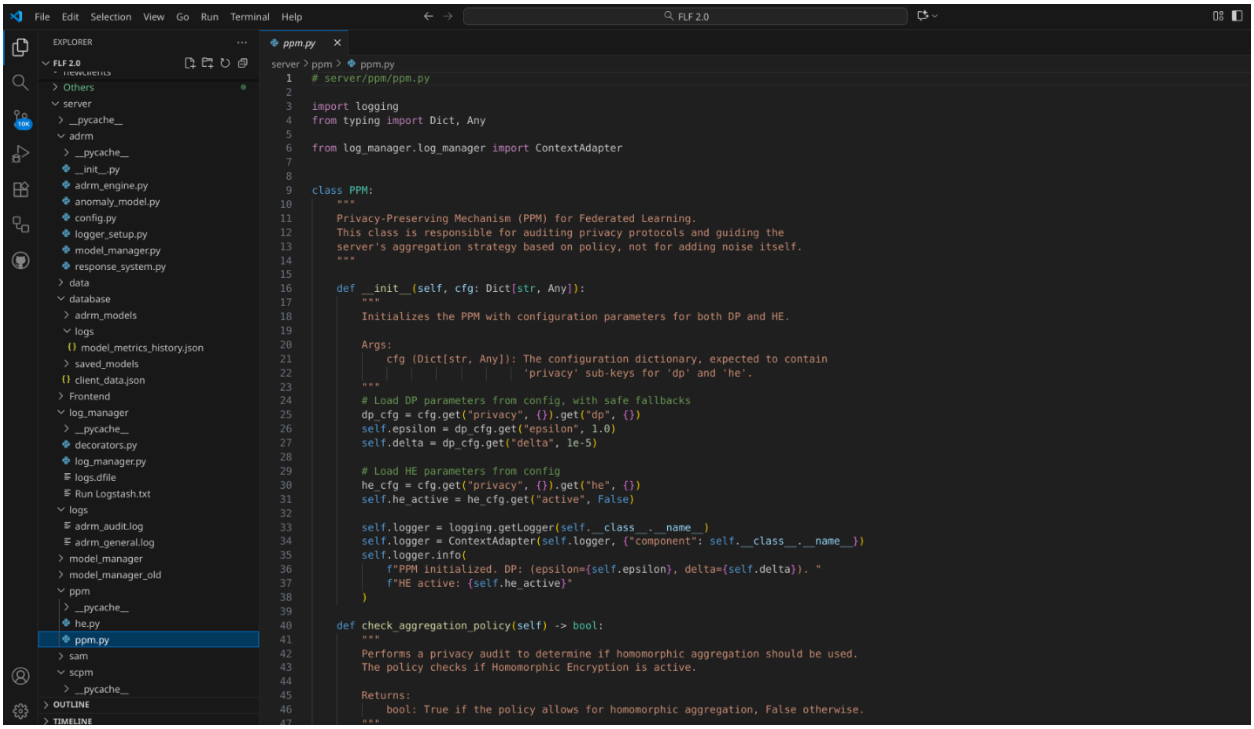


6.3 System Testing



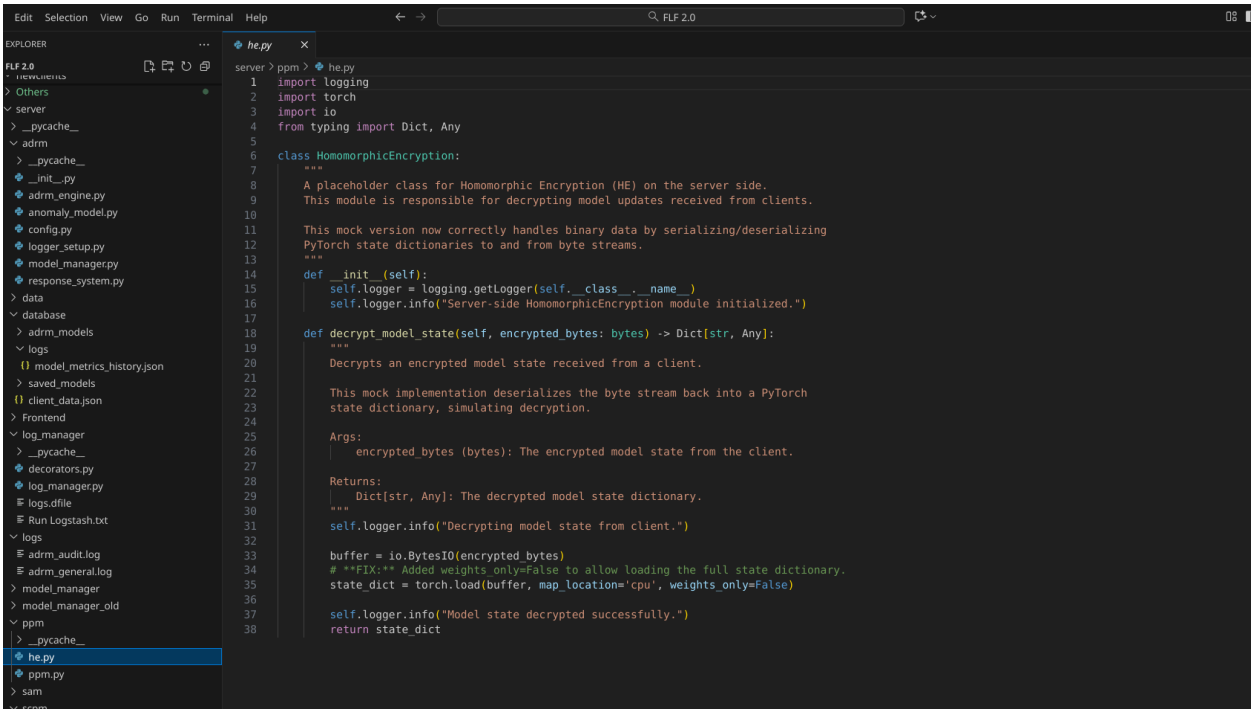
6.4 System Codes

PPM – Server



```python
# server/ppm/ppm.py

import logging
from typing import Dict, Any

from log_manager.log_manager import ContextAdapter


class PPM:
    """
    Privacy-Preserving Mechanism (PPM) for Federated Learning.
    This class is responsible for auditing privacy protocols and guiding the
    server's aggregation strategy based on policy, not for adding noise itself.
    """

    def __init__(self, cfg: Dict[str, Any]):
        """
        Initializes the PPM with configuration parameters for both DP and HE.

        Args:
            cfg (Dict[str, Any]): The configuration dictionary, expected to contain
                                  'privacy' sub-keys for 'dp' and 'he'.
        """
        # Load DP parameters from config, with safe fallbacks
        dp_cfg = cfg.get("privacy", {}).get("dp", {})
        self.epsilon = dp_cfg.get("epsilon", 1.0)
        self.delta = dp_cfg.get("delta", 1e-5)

        # Load HE parameters from config
        he_cfg = cfg.get("privacy", {}).get("he", {})
        self.he_active = he_cfg.get("active", False)

        self.logger = logging.getLogger(self.__class__.__name__)
        self.logger = ContextAdapter(self.logger, {"component": self.__class__.__name__})
        self.logger.info(
            f"PPM initialized. DP: (epsilon={self.epsilon}, delta={self.delta}). "
            f"HE active: {self.he_active}"
        )

    def check_aggregation_policy(self) -> bool:
        """
        Performs a privacy audit to determine if homomorphic aggregation should be used.
        The policy checks if Homomorphic Encryption is active.

        Returns:
            bool: True if the policy allows for homomorphic aggregation, False otherwise.
        """
```

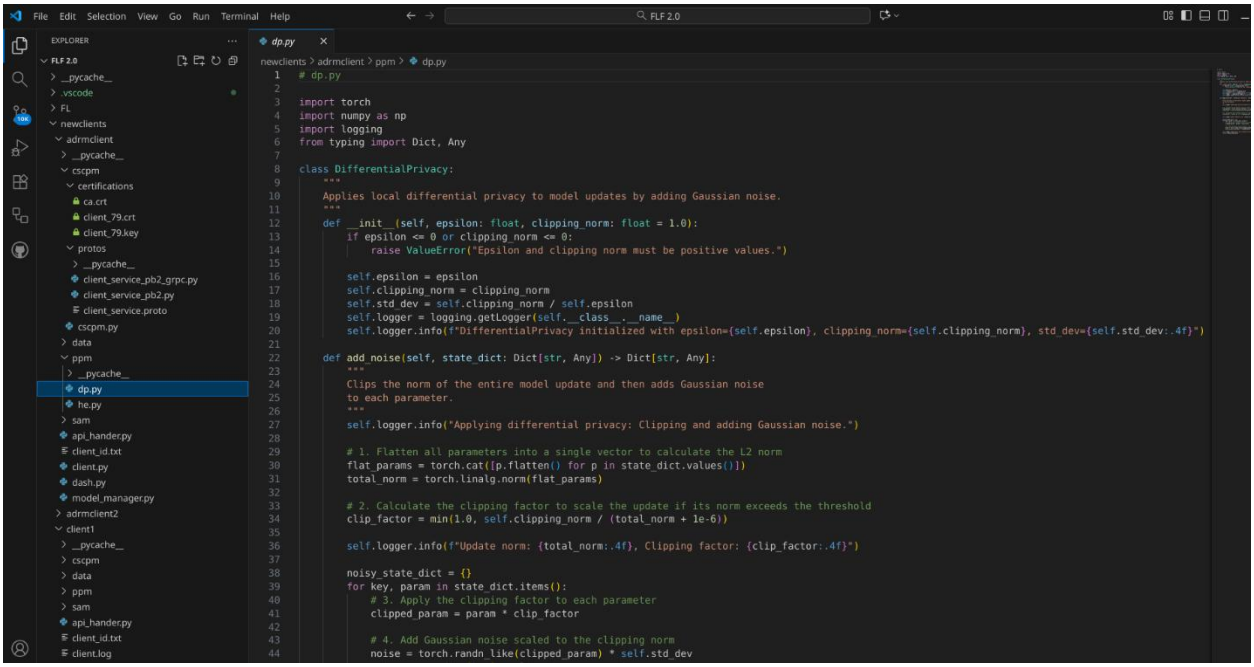
Homomorphic Aggrregation



```python
import logging
import torch
import io
from typing import Dict, Any

class HomomorphicEncryption:
    """
    A placeholder class for Homomorphic Encryption (HE) on the server side.
    This module is responsible for decrypting model updates received from clients.

    This mock version now correctly handles binary data by serializing/deserializing
    PyTorch state dictionaries to and from byte streams.
    """
    def __init__(self):
        self.logger = logging.getLogger(self.__class__.__name__)
        self.logger.info("Server-side HomomorphicEncryption module initialized.")

    def decrypt_model_state(self, encrypted_bytes: bytes) -> Dict[str, Any]:
        """
        Decrypts an encrypted model state received from a client.

        This mock implementation deserializes the byte stream back into a PyTorch
        state dictionary, simulating decryption.

        Args:
            encrypted_bytes (bytes): The encrypted model state from the client.

        Returns:
            Dict[str, Any]: The decrypted model state dictionary.
        """
        self.logger.info("Decrypting model state from client.")

        buffer = io.BytesIO(encrypted_bytes)
        # **FIX:** Added weights_only=False to allow loading the full state dictionary.
        state_dict = torch.load(buffer, map_location='cpu', weights_only=False)

        self.logger.info("Model state decrypted successfully.")
        return state_dict
```

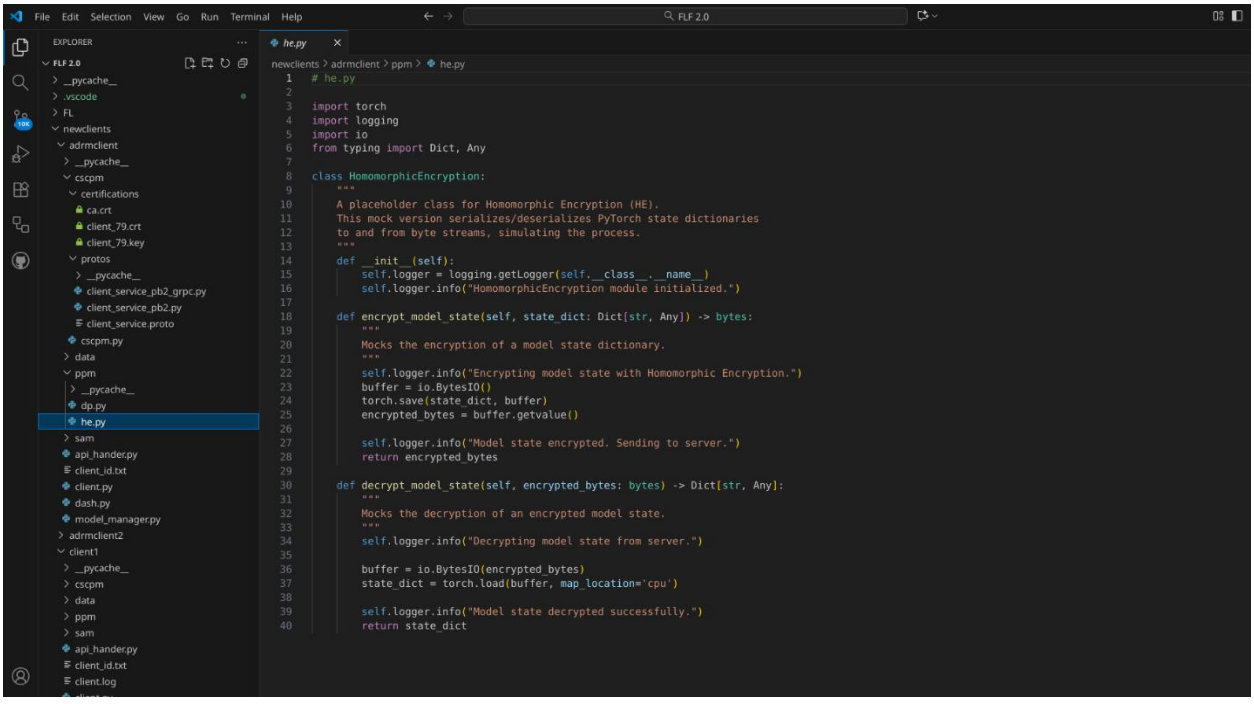
PPM – Clients

Differential Privacy



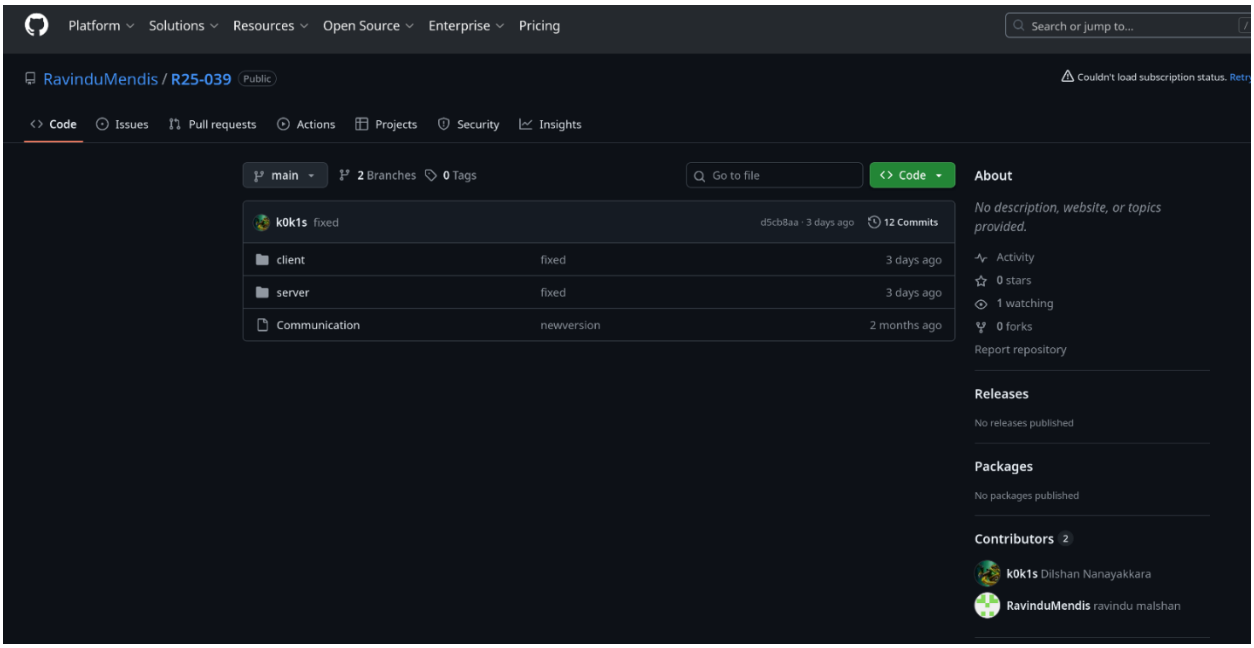
```python
# dp.py

import torch
import numpy as np
import logging
from typing import Dict, Any

class DifferentialPrivacy:
    """
    Applies local differential privacy to model updates by adding Gaussian noise.
    """
    def __init__(self, epsilon: float, clipping_norm: float = 1.0):
        if epsilon <= 0 or clipping_norm <= 0:
            raise ValueError("Epsilon and clipping norm must be positive values.")

        self.epsilon = epsilon
        self.clipping_norm = clipping_norm
        self.std_dev = self.clipping_norm / self.epsilon
        self.logger = logging.getLogger(self.__class__.__name__)
        self.logger.info(f"DifferentialPrivacy initialized with epsilon={self.epsilon}, clipping_norm={self.clipping_norm}, std_dev={self.std_dev:.4f}")

    def add_noise(self, state_dict: Dict[str, Any]) -> Dict[str, Any]:
        """
        Clips the norm of the entire model update and then adds Gaussian noise
        to each parameter.
        """
        self.logger.info("Applying differential privacy: Clipping and adding Gaussian noise.")

        # 1. Flatten all parameters into a single vector to calculate the L2 norm
        flat_params = torch.cat([p.flatten() for p in state_dict.values()])
        total_norm = torch.linalg.norm(flat_params)

        # 2. Calculate the clipping factor to scale the update if its norm exceeds the threshold
        clip_factor = min(1.0, self.clipping_norm / (total_norm + 1e-6))

        self.logger.info(f"Update norm: {total_norm:.4f}, Clipping factor: {clip_factor:.4f}")

        noisy_state_dict = {}
        for key, param in state_dict.items():
            # 3. Apply the clipping factor to each parameter
            clipped_param = param * clip_factor

            # 4. Add Gaussian noise scaled to the clipping norm
            noise = torch.randn_like(clipped_param) * self.std_dev
```

Homomorphic Encryption



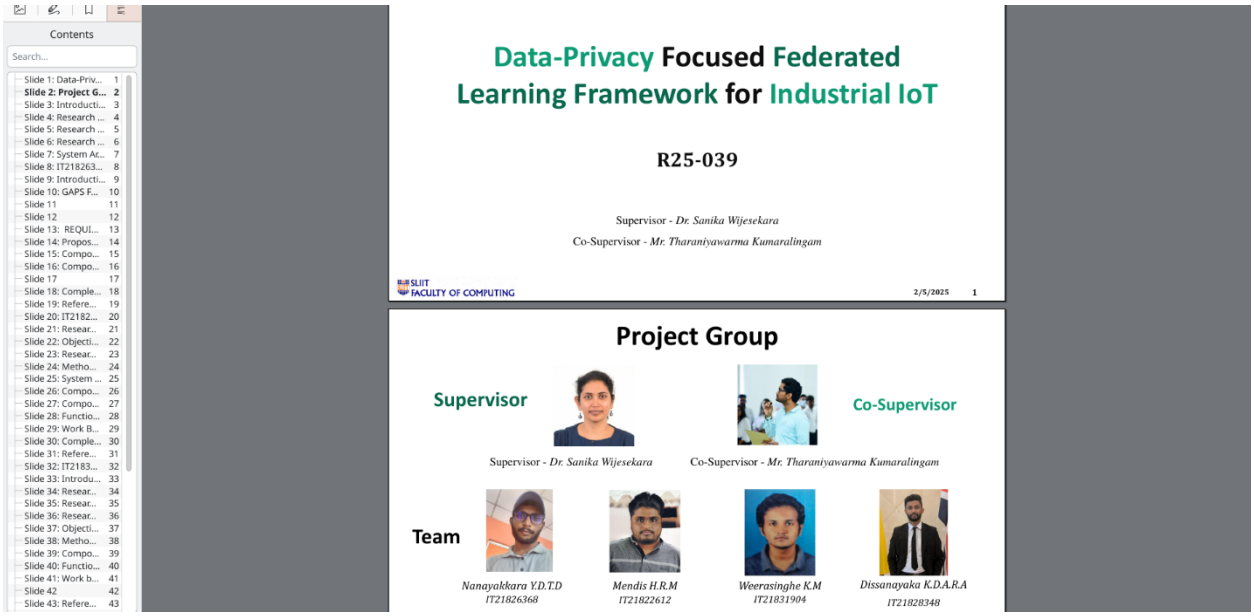
```python
# he.py

import torch
import logging
import io
from typing import Dict, Any

class HomomorphicEncryption:
    """
    A placeholder class for Homomorphic Encryption (HE).
    This mock version serializes/deserializes PyTorch state dictionaries
    to and from byte streams, simulating the process.
    """
    def __init__(self):
        self.logger = logging.getLogger(self.__class__.__name__)
        self.logger.info("HomomorphicEncryption module initialized.")

    def encrypt_model_state(self, state_dict: Dict[str, Any]) -> bytes:
        """
        Mocks the encryption of a model state dictionary.
        """
        self.logger.info("Encrypting model state with Homomorphic Encryption.")
        buffer = io.BytesIO()
        torch.save(state_dict, buffer)
        encrypted_bytes = buffer.getvalue()

        self.logger.info("Model state encrypted. Sending to server.")
        return encrypted_bytes

    def decrypt_model_state(self, encrypted_bytes: bytes) -> Dict[str, Any]:
        """
        Mocks the decryption of an encrypted model state.
        """
        self.logger.info("Decrypting model state from server.")

        buffer = io.BytesIO(encrypted_bytes)
        state_dict = torch.load(buffer, map_location='cpu')

        self.logger.info("Model state decrypted successfully.")
        return state_dict
```

7. GitHub Upload

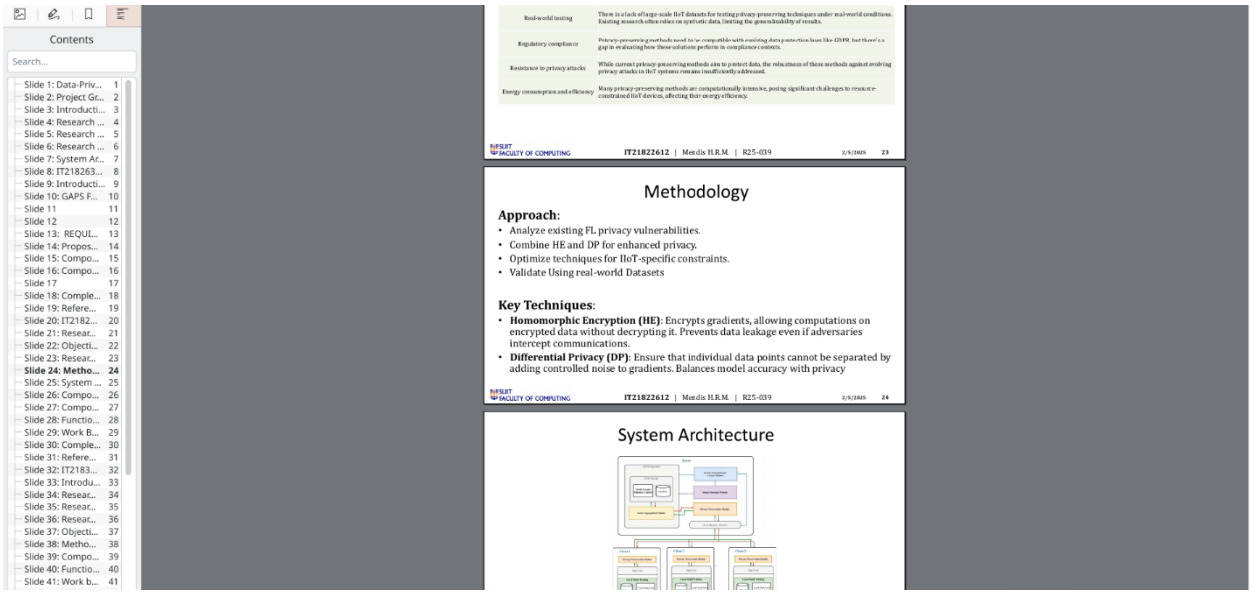


8. Documentation

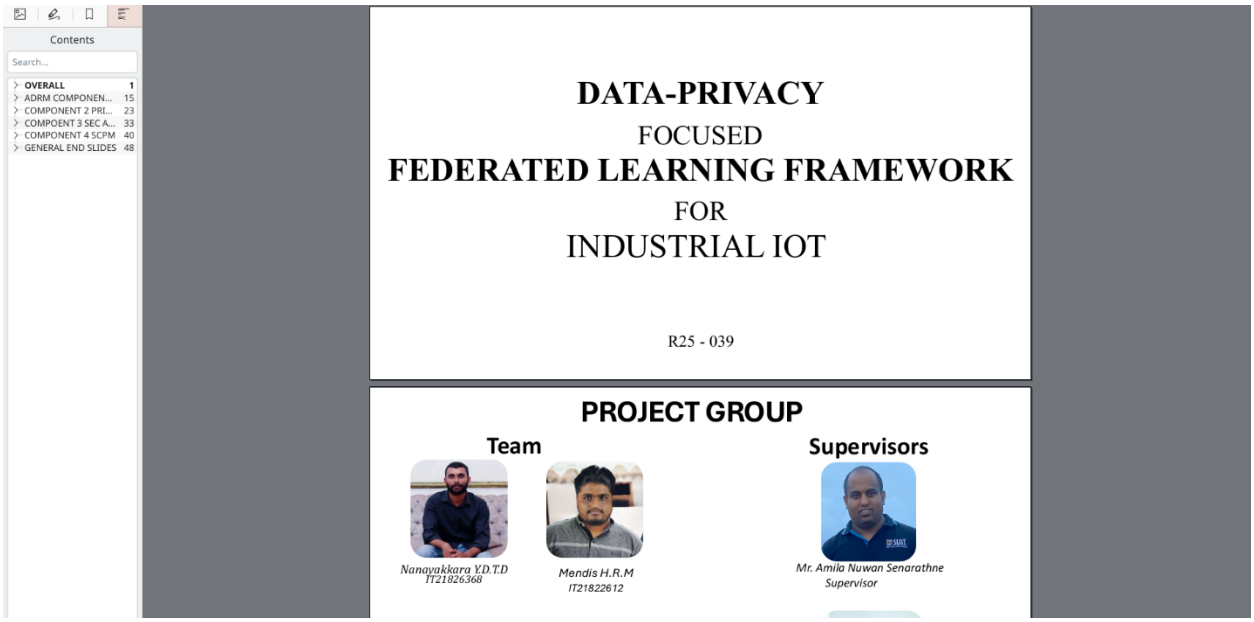
8.1 Proposal



8.2 Presentation 1



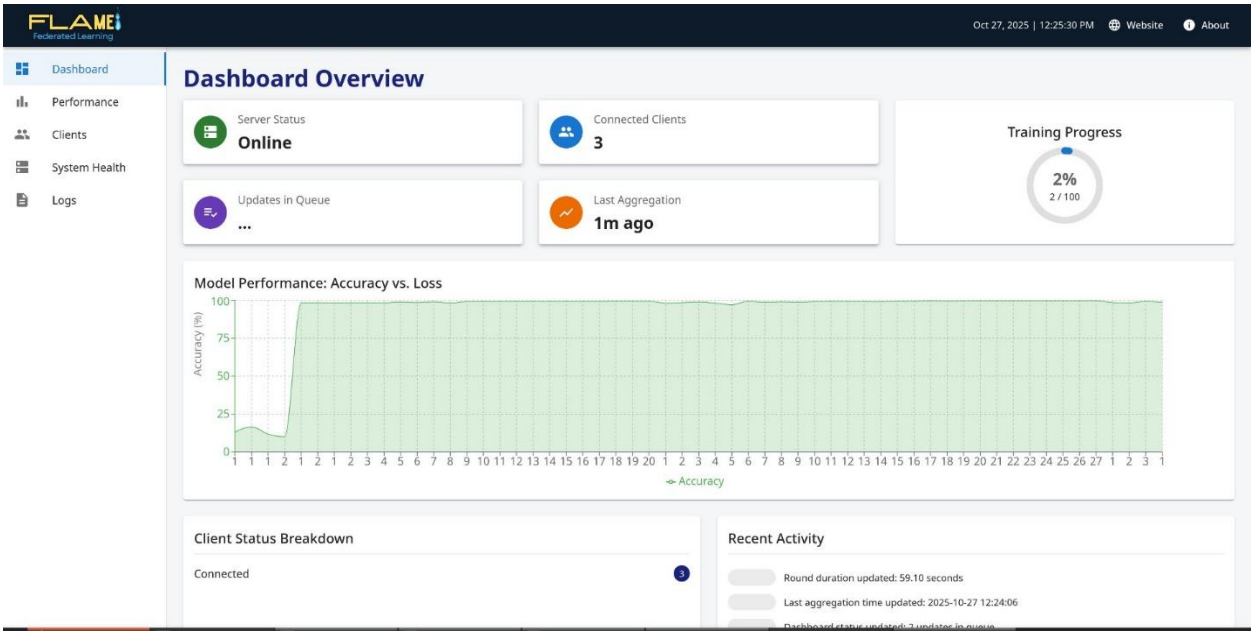
8.3 Presentation 2



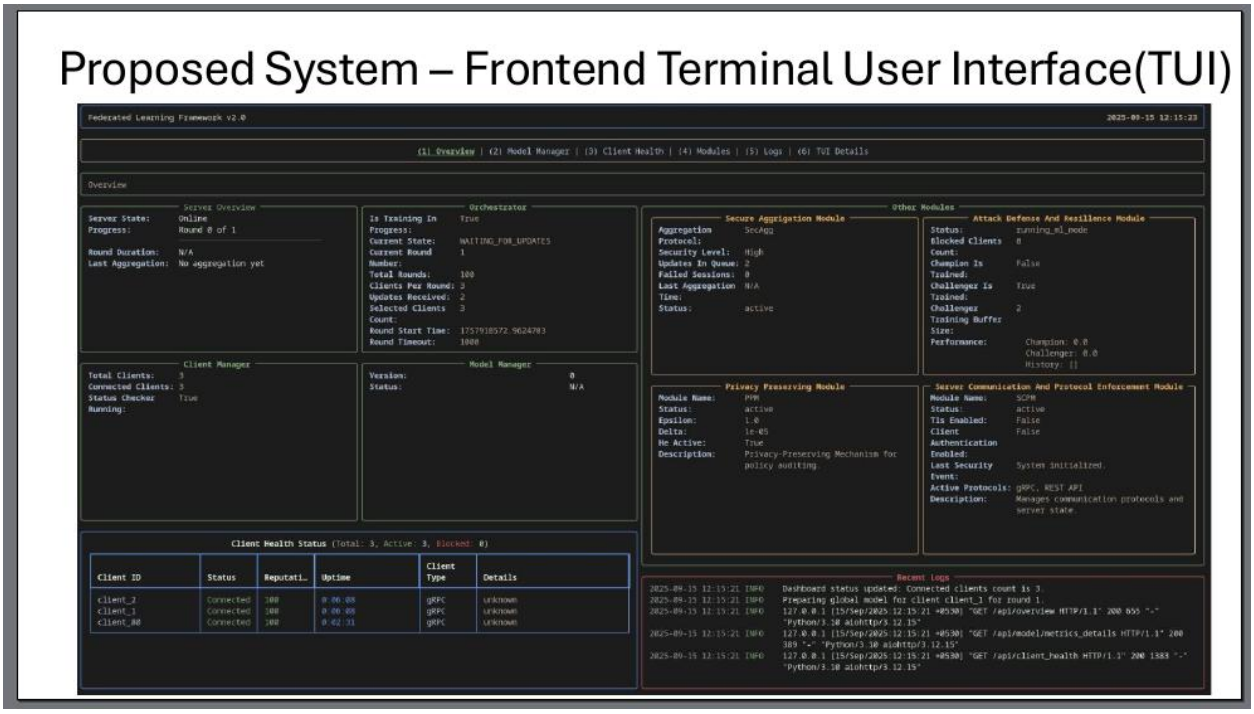
8.4 Final Presentation

8.4 Final Product

Web Portal Frontend



Terminal User interface (Frontend)



8.5 Research Paper

I. Conference Appetence

To Tharindu D Nanayakkara <dilz.nanayakkara@gmail.com> 10/29/25, 11:27 AM

Acceptance Notification

Dear Tharindu D Nanayakkara,

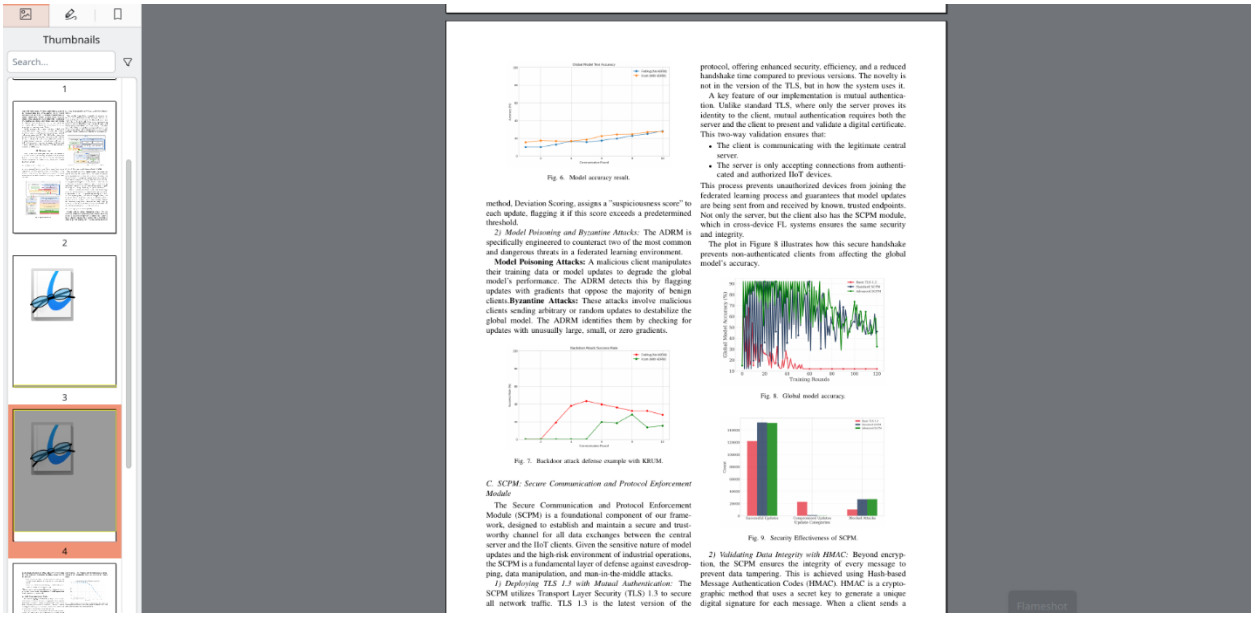
Congratulations! We are pleased to inform you that your paper has been accepted as a regular paper to be presented at the 7th International Conference on Advancements in Computing 2025.

Paper ID: 469
Paper Title: Data-privacy based Federated Learning Framework for Industrial IOT

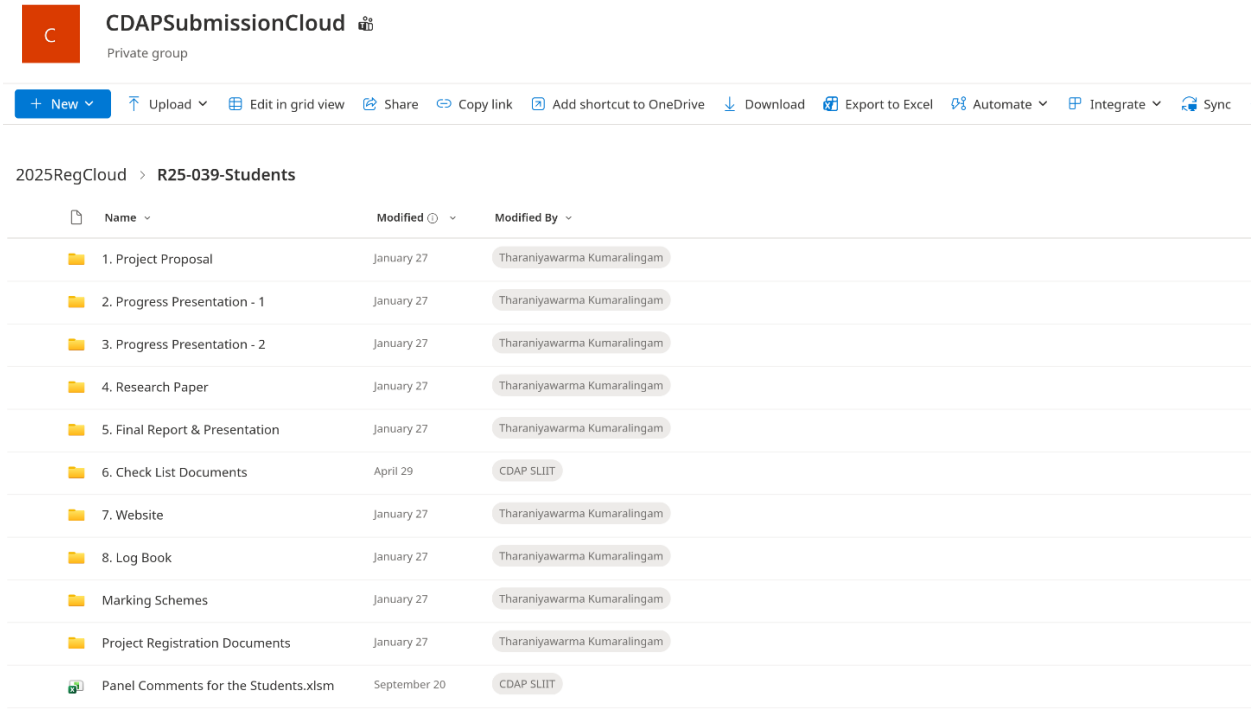
Please visit https://cmt3.research.microsoft.com/7ICAC2025/Submission/Index to view the reviews given during the double-blind review process.

When preparing the camera-ready version of your paper, please address all the review comments and follow the camera-ready guidelines given in the https://icac.lk/for-authors

Please note that the camera-ready deadline is 10th November 2025 and camera-ready submission portal on CMT will be available starting from 22nd October 2025.

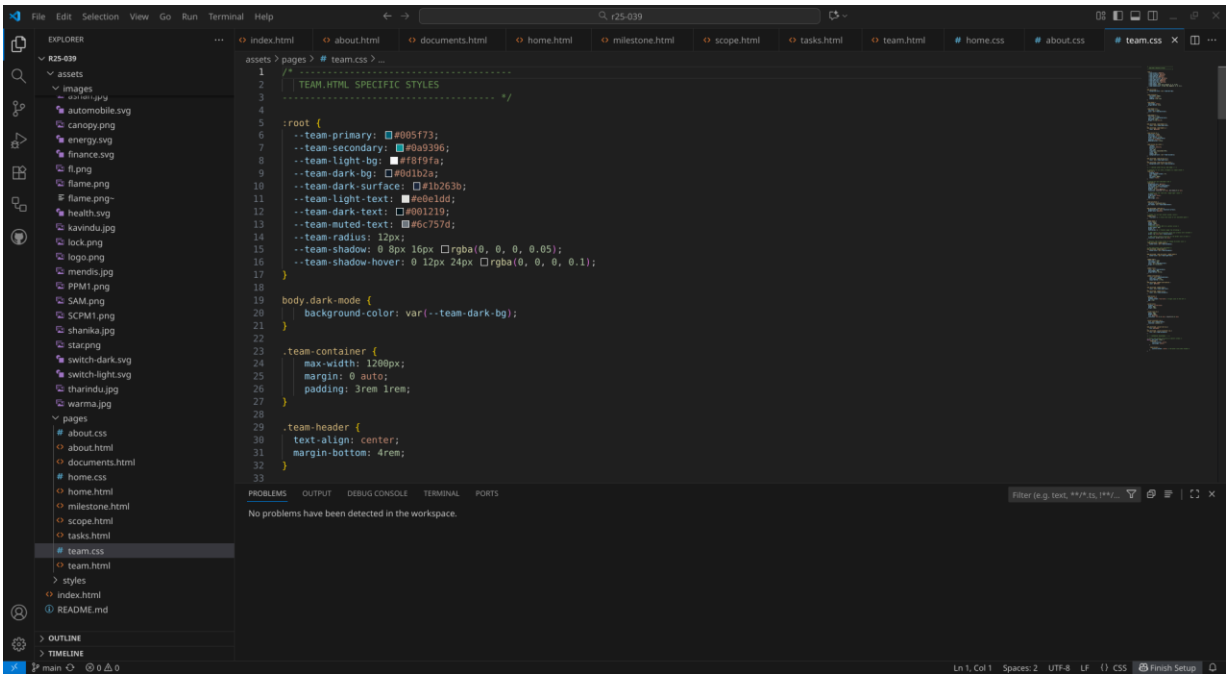


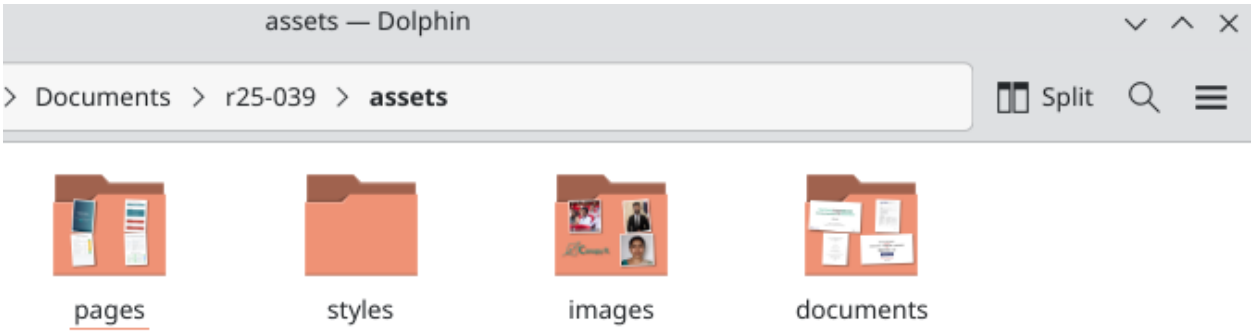
9. CDAP upload

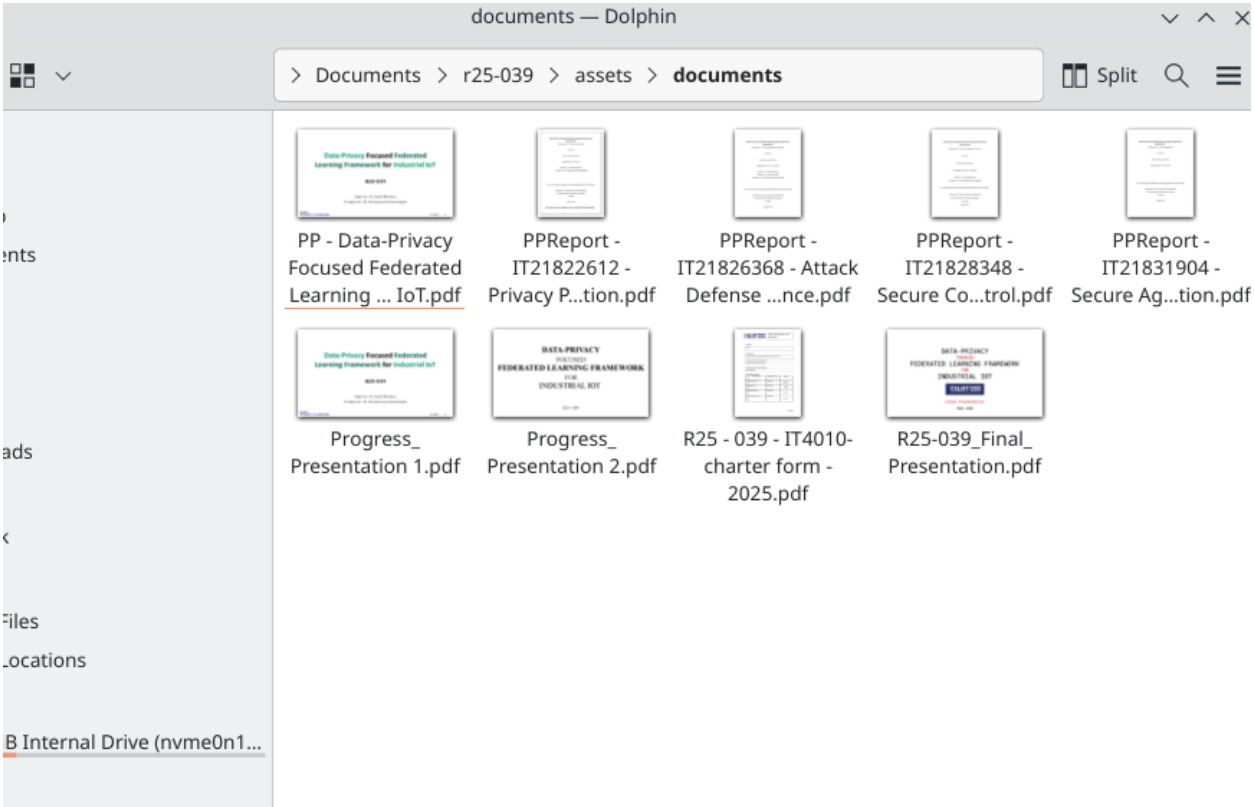


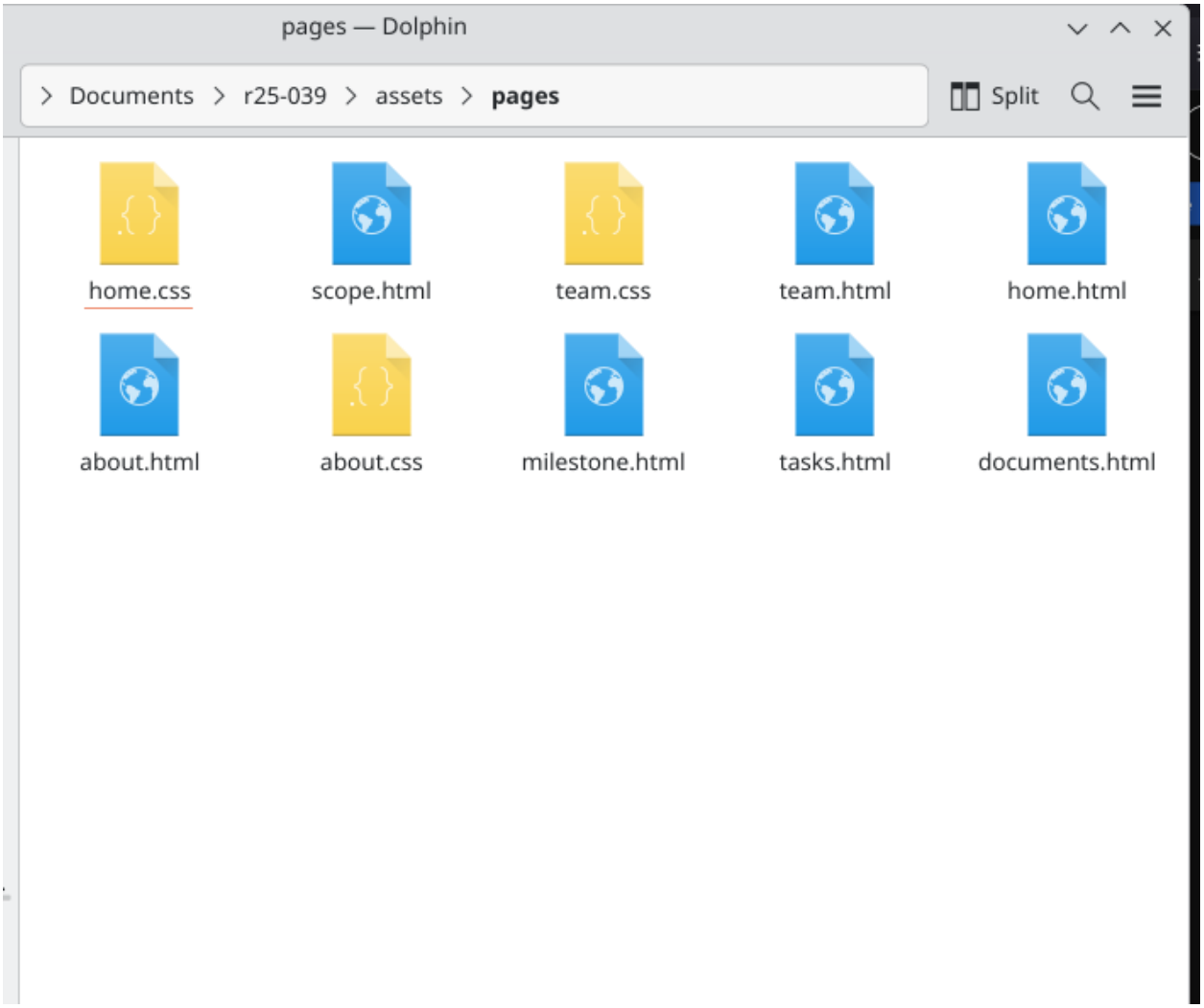
10. Website

10.1 Development

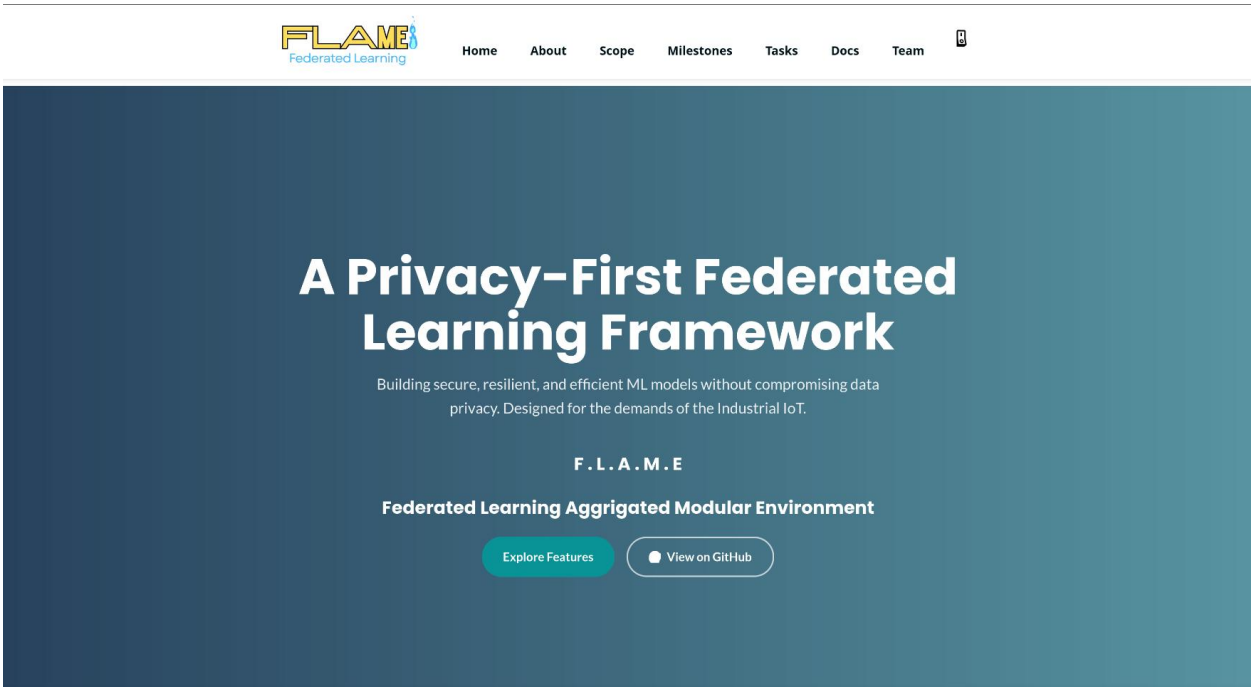



assets — Dolphin
Documents
r25-039
assets
Split
pages
styles
images
documents


documents — Dolphin
Documents
r25-039
assets
documents
Split
PP - Data-Privacy Focused Federated Learning ... IoT.pdf
PPReport - IT21822612 - Privacy P...tion.pdf
PPReport - IT21826368 - Attack Defense ...nce.pdf
PPReport - IT21828348 - Secure Co...trol.pdf
PPReport - IT21831904 - Secure Ag...tion.pdf
Progress_ Presentation 1.pdf
Progress_ Presentation 2.pdf
R25 - 039 - IT4010-charter form - 2025.pdf
R25-039_Final_ Presentation.pdf


pages — Dolphin
Documents
r25-039
assets
pages
Split
home.css
scope.html
team.css
team.html
home.html
about.html
about.css
milestone.html
tasks.html
documents.html

10.2 Finalize



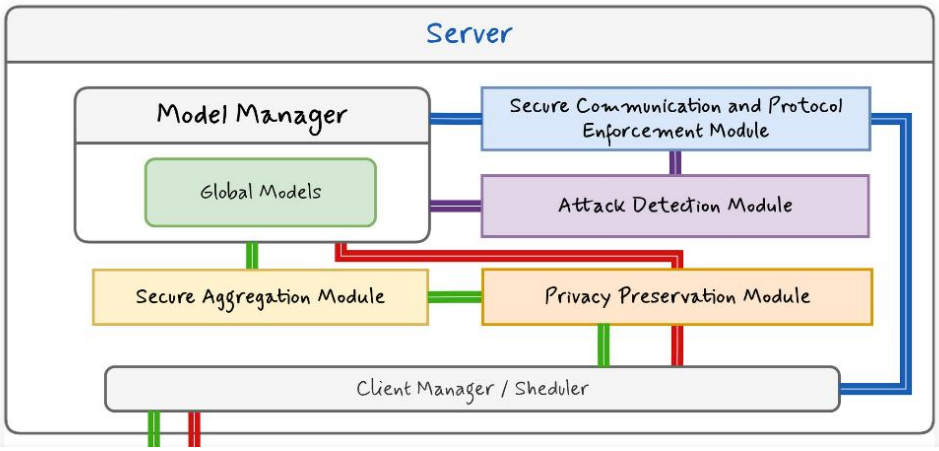
FLAME Federated Learning

Home | About | Scope | Milestones | Tasks | Docs | Team

Privacy-Enhanced Federated Learning Framework

Our comprehensive Federated Learning (FL) System Framework is engineered to significantly augment the privacy, security, and operational resilience of machine learning models deployed in decentralized and distributed environments. The framework is composed of four interconnected core modules, collectively guaranteeing data integrity, defense against adversarial attacks, and authenticated inter-component communication.

Framework Overview and Architecture



FLAME Federated Learning

Home About Scope Milestones Tasks Docs Team

Mr. Amila Senerathne

Supervisor

Dr. Sanika Wijesekara

External Supervisor

Mr. T. Kumaralingam

Co-Supervisor