APPENDIX

I. USER INTERFACE

To make the system interactive so that the end-users including farmers and local stakeholders can upload their data and receive irrigation decision, we have designed a user interface. The user has to first register using the user interface by providing his/her details such as Name, Contact number, Email ID, etc., as presented in Figure 1. The user also needs to provide the Password he/she wants to set for further login. After providing the details, the user clicks on the 'Register' button. After successful registration, the user can login to the system by providing the Email ID and password as the login credentials, as shown in Figure 2. After successful login the user can browse file and select the file to upload, as presented in Figure 3. Usually, the IoT data are stored in a CSV file. Thus, in the figure the user selects a CSV file containing the soil and environmental data. After uploading the file, the user needs to wait for the decision. The data is extracted from the file by the edge server, the model predicts the result for the input data, and the result is displayed on the screen of the user's device as shown in Figure 4. As we observe, the user receives the decision that 'Irrigation is required'.

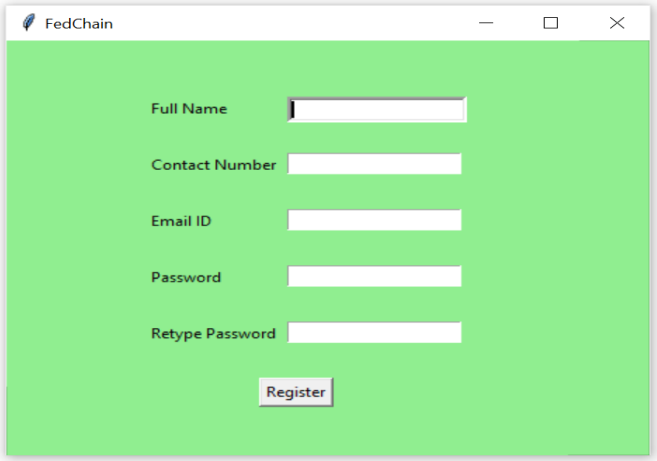


Fig. 1: Registration Page

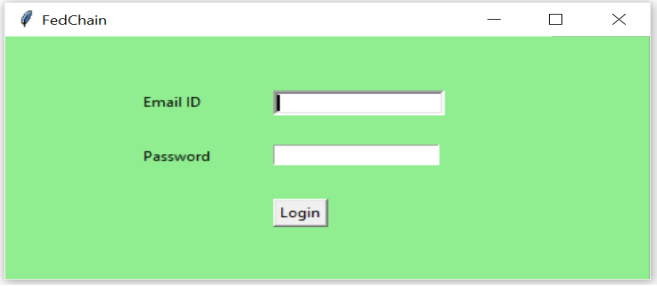


Fig. 2: Login Page

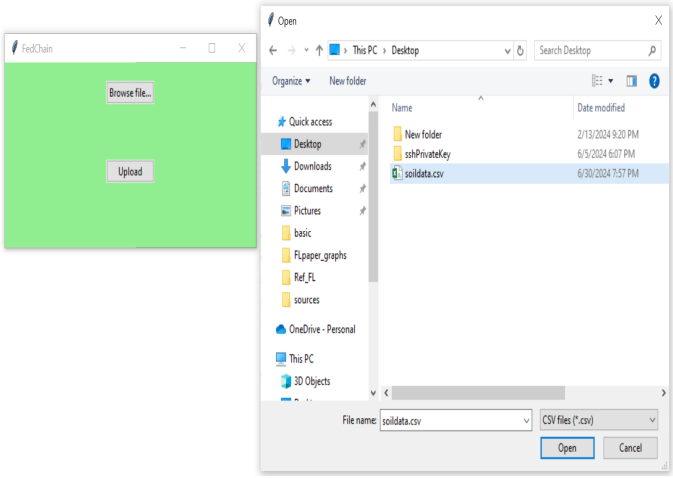


Fig. 3: File upload Page

Fig. 4: Receive Irrigation Decision

The entire scenario is depicted in Figure 5.

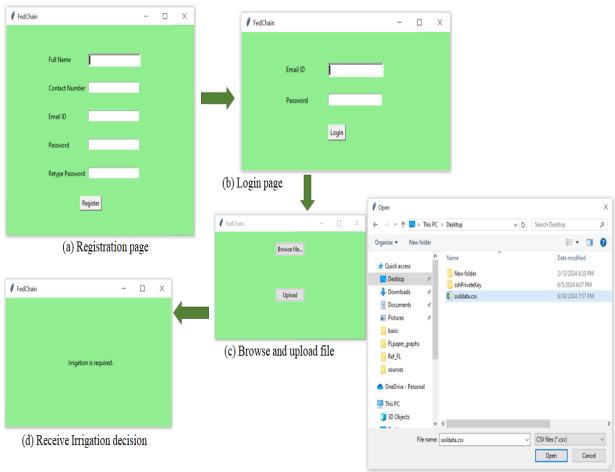


Fig. 5: Designed User interface for registration, login, upload file, and receive decision

II. COMPUTATIONAL COMPLEXITY ANALYSIS

The FedChain has several operations with different time, space, and computational complexity, which are summarized in TABLE I and discussed as follows:

- The time complexity for model initialization in each peer is $O(1)$. However, the space complexity for initial model

TABLE I: Time, space, and computational complexity of different operations in FedChain

Operation	Time Complexity	Space Complexity	Overall Computational Complexity
Model initialization	$O(1)$	$O(m_p)$	$O(m_p)$
Data Loading	$O(\|D_p\|)$	$O(\|D_p\|)$	$O(\|D_p\|)$
Label Encoding	$O(\|D_p\|)$	$O(\|D_p\|)$	$O(\|D_p\|)$
Standard Scaling	$O(\|D_p\|)$	$O(\|D_p\|)$	$O(\|D_p\|)$
Train-Test Split	$O(\|D_p\|)$	$O(\|D_p\|)$	$O(\|D_p\|)$
Training	$O(R \cdot \eta \cdot N_B \cdot m_p)$	$O(m_p)$	$O(R \cdot \eta \cdot N_B \cdot m_p)$
Aggregation	$O(R \cdot \|M\| \cdot m_j)$	$O(\|M\| \cdot m_j)$	$O(R \cdot \|M\| \cdot m_j)$
IPFS Store	$O(\|\Delta m\|)$	$O(\|\Delta m\|)$	$O(\|\Delta m\|)$
DHT Update	$O(\log \|P\|)$	$O(1)$	$O(\log \|P\|)$
Digital Signing	$O(1)$	$O(1)$	$O(1)$
Broadcast Transaction	$O(1)$	$O(1)$	$O(1)$
SC Validation	$O(1)$	$O(1)$	$O(1)$
Commit Transaction	$O(1)$	$O(1)$	$O(1)$

creation depends on the model parameters (m_p). Hence, the space complexity for initial model creation is $O(m_p)$. The overall computational complexity for initial model creation is also $O(m_p)$.

- Each peer p has its own local dataset (D_p). The time complexity, space complexity, and overall computational complexity for data loading depend on the size of the dataset, hence, $O(|D_p|)$. The label encoding, standard scaling, and train-test split, also have the time, space, and overall computational complexity of $O(|D_p|)$.
- The time complexity for model training in each peer depends on the number of rounds (R), number of epochs (η), size of the dataset (D_p), batch size (B), and model parameters (m_p). Thus, in FedChain, the time complexity for model training is given as $O(R \cdot \eta \cdot N_B \cdot m_p)$, where $N_B = |D_p|/B$. The space complexity for model training in FedChain is given as $O(m_p)$. Finally, the overall computational complexity for model training in FedChain is given as $O(R \cdot \eta \cdot N_B \cdot m_p)$.
- The time complexity for aggregation depends on the number of neighbour nodes ($|M|$, where M is the set of neighbour nodes), received model parameters (m_j from neighbour j, $j \in M$), and the number of rounds (R). Thus, for aggregation in FedChain, the time complexity is $O(R \cdot |M| \cdot m_j)$, the space complexity is $O(|M| \cdot m_j)$, and the overall computational complexity is $O(R \cdot |M| \cdot m_j)$, where m_j denotes model parameters received from neighbour j ($j \in M$).
- The time, space, and overall computational complexity for IPFS Store is of $O(|\Delta m|)$, where Δm denotes model update in the decentralized federated learning. For DHT Update, the time complexity is $O(\log |P|)$, space complexity is $O(1)$, and overall computational complexity is $O(\log |P|)$, where P is the set of peers in the P2P network formed by the edge servers. The time, space, and overall computational complexity for Digital Signing, broadcast transaction, SC validation, and commit transaction are of $O(1)$.

III. Blockchain Details and Implementation

For our proposed system, we have selected a permissioned Ethereum blockchain due to its controlled access, enhanced security, and optimized performance suitable for our decentralized federated learning (DFL) framework. In this permissioned blockchain, only authorized nodes, denoted as N = $\{n_1, n_2, \ldots, n_k\}$, where n_i represents a node in the network, are allowed to participate. Each node in the blockchain network plays a specific role, such as a validator or a participant, ensuring the integrity and security of the network. The nodes follow a Proof-of-Stake (PoS) consensus mechanism, which is more efficient and suitable for permissioned environments compared to Proof of Work (PoW).

In our implementation, each local client (peer) trains its model on local data and stores the resulting model updates in the InterPlanetary File System (IPFS). The IPFS returns a unique Content Identifier (CID) for each stored update, denoted as ϕ_{CID}. The client then generates a blockchain transaction containing the ϕ_{CID}, the peer ID p_{id}, and a timestamp T. This transaction is digitally signed using the client's private key to ensure authenticity and prevent tampering. The signed transaction τ is then submitted to the Ethereum network. The smart contract, deployed on the Ethereum blockchain using Remix, verifies the digital signature and checks if the peer is authorized to submit the update. Upon successful validation, the smart contract records the transaction on the blockchain, ensuring that the CID and metadata are immutably stored.

The process of transaction ordering begins with the submission of transactions by peers to their respective Ethereum nodes. Each transaction, including the CID from IPFS and the signed metadata, is validated by the nodes according to the rules defined in the smart contract. Valid transactions are then included in the next block by the validators. The blocks are created and propagated across the network, ensuring all participating nodes have a consistent view of the blockchain.

The block header fields in our blockchain include the block number $Block_{num}$, the hash of the previous block H_{prev}, the Merkle root H_{merkle}, the timestamp T_{block}, the nonce N_{block}, the miner ID $Miner_{id}$, the gas limit Gas_{limit}, and the gas used Gas_{used}. This structure ensures data integrity and traceability of all transactions.

The experimental setup involves configuring a permissioned Ethereum network with multiple validator nodes, utilizing Docker containers for consistency and scalability. The network is designed to use the PoA consensus mechanism to facilitate efficient block validation and creation. A client application is developed using Node.js and web3.js to handle model training,

IPFS storage, transaction creation, and submission. The smart contracts, implemented in Solidity, manage the submission and validation of model updates, as well as authorization control. The smart contracts are deployed and interacted with using the Remix IDE. IPFS is integrated with the local client to handle data storage and retrieval efficiently. The experimental setup is designed to evaluate performance metrics such as throughput, latency, and energy consumption, ensuring that the system is secure, scalable, and efficient for decentralized federated learning in smart irrigation systems.

A. Experimental Implementation

- *Install Geth (Go Ethereum)*:
  sudo apt-get install software-properties-common
  sudo add-apt-repository -y ppa:ethereum/ethereum
  sudo apt-get update
  sudo apt-get install ethereum
- *Initialize the Blockchain*:
  geth –datadir node1 init genesis.json
  geth –datadir node2 init genesis.json

- *Create Validator Accounts*:
  geth –datadir node1 account new
  geth –datadir node2 account new

- *Start Blockchain nodes and Connect*:
  geth –datadir node1 –networkid 1234 –nodiscover –http –http.port 8545 –http.addr ”127.0.0.1” –port 30303 –allow-insecure-unlock –unlock ”0x1234567890abcdef1234567890abcdef12345678” –mine –miner.threads 1 –http.corsdomain ”*” –http.api personal,db,eth,net,web3,txpool,miner –password node1/password.txt
  geth –datadir node2 –networkid 1234 –nodiscover –http –http.port 8546 –http.addr ”127.0.0.1” –port 30304 –allow-insecure-unlock –unlock ”0xabcdef1234567890abcdef1234567890abcdef12” –mine –miner.threads 1 –http.corsdomain ”*” –http.api personal,db,eth,net,web3,txpool,miner –password node2/password.txt
  geth attach http://127.0.0.1:8545
  admin.addPeer(”enode://abcd1234@127.0.0.1:30304”)
- *Installing IPFS*:
  wget https://dist.ipfs.io/go-ipfs/v0.8.0/go-ipfs_v0.8.0_linux-amd64.tar.gz
  tar xvfz go-ipfs_v0.8.0_linux-amd64.tar.gz cd go-ipfs
  sudo bash install.sh
  ipfs init
  ipfs daemon

- *Store Model Updates in IPFS*:
  const ipfsClient = require(’ipfs-http-client’);
  const ipfs = ipfsClient(host: ’localhost’, port: ’5001’, protocol: ’http’);
  async function storeModelUpdate(modelUpdate)
  const cid = await ipfs.add(modelUpdate);
  return cid.toString();

- *Client Application (Node.js)*:
  const Web3 = require(’web3’);
  const web3 = new Web3(’http://localhost:8545’);
  const ModelStorage = require(’./DFLAccessControl.json’);
  #ABI of the deployed contract (FedChain.sol)
  const contractAddress = ’0x0x1234567890abcdef1234567890abcdef12345678’;
  const modelStorage = new web3.eth.Contract(ModelStorage.abi, contractAddress);

  async function submitModelUpdate(modelUpdate, account)
  const cid = await storeModelUpdate(modelUpdate);
  const updateId = web3.utils.sha3(cid + account); // Create a unique update ID
  await modelStorage.methods.addModelUpdate(updateId, cid).send(from: account);
  console.log(‘Model update stored with CID: cid‘);
  async function getModelUpdate(updateId)
  const cid = await modelStorage.methods.getModelUpdateCID(updateId).call();
  console.log(‘Model CID: cid‘);

- *Running Client*:
  node client.js

IV. Convergence Analysis of DFL in *FedChain*

In the proposed *FedChain* framework, we consider a set of peer nodes P. Each peer node p ($p \in P$) has a local dataset D_p, and maintains a local model with parameters m_p. The objective is to minimize the global loss function $L(m)$, which is defined as the weighted average of the local loss functions $L_p(m)$ at each node, presented as follows.

$$L(m) = \frac{1}{K} \sum_{p=1}^{K} L_p(m) \tag{1}$$

where $L_p(m)$ is the local loss function at node p and $K = |P|$.

Definition IV.1. *Lipschitz Continuity: The local loss function $L_p(m)$ is L-Lipschitz continuous if there exists a constant $Z > 0$ such that $\forall m, \mathbf{v}$, we have*

$$L_p(m) \leq L_p(\mathbf{v}) + \nabla L_p(\mathbf{v})^T (m - \mathbf{v}) + \frac{Z}{2} \|m - \mathbf{v}\|^2 \tag{2}$$

Definition IV.2. *Bounded Variance: The variance of the stochastic gradients is bounded if there exists a constant $C_1^2 > 0$ such that $\forall m$:*

$$\mathbb{E}[\|\nabla L_p(m) - \nabla L(m)\|^2] \leq C_1^2 \tag{3}$$

Definition IV.3. *Unbiased Gradients: The stochastic gradients are unbiased estimates of the true gradients if $\forall m$:*

$$\mathbb{E}[\nabla L_p(m)] = \nabla L(m) \tag{4}$$

Assumption IV.1. *Smoothness: The global loss function $L(m)$ is smooth, i.e., there exists a constant $C_2 > 0$, such that $\forall m, \mathbf{v}$:*

$$L(m) \leq L(\mathbf{v}) + \nabla L(\mathbf{v})^T (m - \mathbf{v}) + \frac{C_2}{2} \|m - \mathbf{v}\|^2 \tag{5}$$

Assumption IV.2. *Strong Convexity: The global loss function $L(m)$ is strongly convex, i.e., there exists a constant $C_3 > 0$ such that $\forall m, \mathbf{v}$:*

$$L(m) \geq L(\mathbf{v}) + \nabla L(\mathbf{v})^T (m - \mathbf{v}) + \frac{C_3}{2} \|m - \mathbf{v}\|^2 \quad (6)$$

Lemma IV.1. *Gradient Bound Under the assumptions of Lipschitz continuity and bounded variance, the gradient of the global loss function $L(m)$ is bounded:*

$$\mathbb{E}[\|\nabla L(m)\|^2] \leq \frac{Z}{K} \sum_{p=1}^{K} \|\nabla L_p(m)\|^2 + \frac{C_1^2}{K} \quad (7)$$

Proof. By Lipschitz continuity of $L_p(m)$:

$$\|\nabla L_p(m)\|^2 \leq Z^2 \|m\|^2 \quad (8)$$

Taking the expectation and summing over all nodes:

$$\begin{aligned} \mathbb{E}[\|\nabla L(m)\|^2] &= \frac{1}{K^2} \sum_{p=1}^{K} \mathbb{E}[\|\nabla L_p(m)\|^2] \\ &\leq \frac{Z^2}{K^2} \sum_{p=1}^{K} \|m\|^2 + \frac{C_1^2}{K} \end{aligned} \quad (9)$$

Thus,

$$\mathbb{E}[\|\nabla L(m)\|^2] \leq \frac{Z}{K} \sum_{p=1}^{K} \|\nabla L_p(m)\|^2 + \frac{C_1^2}{K} \quad (10)$$

□

Theorem IV.4. *Under the assumptions of Lipschitz continuity, bounded variance, unbiased gradients, smoothness, and strong convexity, the FedChain framework converges to a stationary point of the global loss function $L(m)$.*

Proof. Step 1: Local Update Rule:
Each node p performs local updates using stochastic gradient descent (SGD) for η local epochs. Let $m_p^{t_k}$ denotes the model parameters at node p at epoch k, then

$$m_p^{t_{k+1}} = m_p^{t_k} - \zeta \nabla m_p^{t_k} \quad (11)$$

where ζ is the learning rate, and

$$\nabla m_p^{t_k} = \frac{\partial L(g(x_p, m_p^{t_k}), y_p)}{\partial m_p^{t_k}} \quad (12)$$

where the model parameters $m_p^{t_k}$ are revised using the gradient descent rule, where a fraction of the gradient is subtracted from the current existing parameters, and $g(.)$ is the result after prediction for the input data x_p, L is the loss function, and y_p is the label data of peer p.
Step 2: Aggregation:
After η local epochs, the nodes exchange their model updates with their neighbors in the P2P network and aggregate the updates. The peer model update is defined as:

$$m_p^{r+1} = \frac{1}{|M|} \sum_{j=1}^{|M|} m_j^r \quad (13)$$

where $|M|$ denotes the total number of neighbour nodes, and m_j^r sets as the model parameters received from neighbour node j at round r.
The global model update is defined as:

$$m^{r+1} = \frac{1}{K} \sum_{p=1}^{K} m_p^{r+1} \quad (14)$$

Step 3: Bounding the Global Loss:
Using the smoothness assumption, we can bound the change in the global loss function as follows.

$$\begin{aligned} L(m^{r+1}) \leq L(m^r) + \nabla L(m^r)^R (m^{r+1} - m^r) \\ + \frac{C_2}{2} \|m^{r+1} - m^r\|^2 \end{aligned} \quad (15)$$

Taking the expectation over the stochastic gradients, we obtain.

$$\mathbb{E}[L(m^{r+1})] \leq L(m^r) - \frac{\zeta}{K} \|\nabla L(m^r)\|^2 + \frac{C_2 \zeta^2 C_1^2}{2K} \quad (16)$$

Now, summing this inequality over R rounds, we get.

$$\sum_{r=1}^{R} \mathbb{E}[L(m^{r+1}) - L(m^r)] \leq -\frac{\zeta}{K} \sum_{r=1}^{R} \|\nabla L(m^r)\|^2 + \frac{C_2 \zeta^2 C_1^2 R}{2K} \quad (17)$$

After rearranging terms, we find

$$\frac{1}{R} \sum_{r=1}^{R} \mathbb{E}[\|\nabla L(m^r)\|^2] \leq \frac{L(m^1) - L(m^{R+1})}{\zeta R} + \frac{C_2 \zeta C_1^2}{2K} \quad (18)$$

As $R \to \infty$, the term $\frac{L(m^1) - L(m^{R+1})}{\eta R}$ approaches zero, ensuring the following.

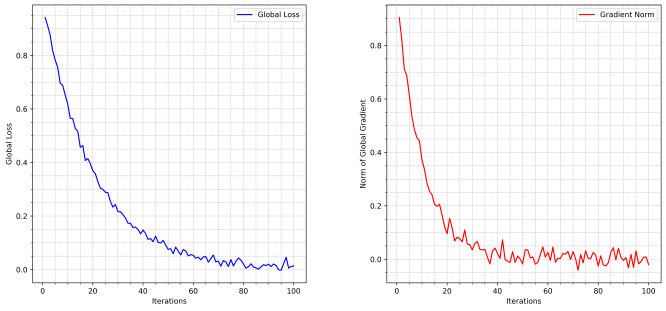
$$\lim_{R \to \infty} \frac{1}{R} \sum_{r=1}^{R} \mathbb{E}[\|\nabla L(m^r)\|^2] = 0 \quad (19)$$

This demonstrates that the global model parameters m converge to a stationary point of the global loss function $L(m)$. □

A. Convergence Plots

In our experiments, we utilized a decentralized setup with $K = 10$ peer nodes. Each node trained its local model using a subset of the soil moisture dataset for 10 local epochs before aggregating the model updates. The learning rate ζ was set to 0.001, and the maximum number of rounds (iterations) R was set to 100. The convergence behavior was analyzed by plotting the global loss function $L(m)$, and the norm of the global gradient $|\nabla L(m)\|$ over the rounds (iterations). Fig. 6 shows the plots.

Fig. 6a illustrates the behavior of the global loss over 100 iterations. The exponential decay observed indicates that the loss function is minimizing as the number of iterations increases, which is a clear sign of convergence. Initially, the global loss starts at a higher value and decreases rapidly, then more gradually as iterations progress, demonstrating the typical behavior of optimization algorithms where rapid initial improvements slow down as the algorithm approaches an optimal solution. The mean global loss decreases from approximately 1.0 at iteration 1 to around 0.01 at iteration 100, with minor fluctuations around the trend line due to added



(a) Convergence of Global Loss Function

(b) Convergence of Global Gradient Norm

Fig. 6: Convergence Behavior

noise representing natural variances in real-world scenarios. This downward trend shows that the DFL algorithm effectively reduces the loss, confirming the theoretical convergence proof provided earlier. The overall behavior of the global loss aligns with our theoretical analysis, where the expected decrease in the global loss function $F(m)$ is guaranteed under the assumptions of the convergence theorem.

Fig. 6b shows the behavior of the norm of the global gradient over the same 100 iterations. The norm of the gradient provides insight into how the steepness of the loss function's slope changes as optimization proceeds. Initially, the gradient norm is relatively high, indicating that the model parameters are far from optimal. As iterations progress, the gradient norm decreases exponentially, stabilizing around a lower value close to zero. The mean gradient norm drops from approximately 1.0 at iteration 1 to around 0.01 at iteration 100, with minor noise-induced fluctuations. The consistent downward trend in the gradient norm demonstrates the statistical significance of this plot, validating that the updates are reducing in magnitude, which is expected as the model approaches convergence. This observation is consistent with our theoretical proof, where the norm of the gradient $|\nabla F(m)\|^2$ is shown to decrease towards zero, ensuring that the model parameters are nearing an optimal solution.