

Leveraging blockchain techniques for predicting and transmitting data over pervasive environment

Aditya Tripathi (202318046), Anurag Choudhury (202318059), Karan Sharma (202318018)
Dr. Rahul Mishra, Dr. Tapas Kumar Maiti

Abstract

The proliferation of Internet of Things (IoT) devices in pervasive environments has necessitated robust solutions for data transmission and predictive analytics. This paper proposes a system that integrates blockchain technology and federated learning to address these challenges. Using the California Housing dataset, the system ensures data privacy, secure model updates, and efficient predictive modeling. Blockchain's decentralized ledger maintains tamper-proof training records, while federated learning enables distributed data processing. Experimental results demonstrate the system's effectiveness in achieving secure, accurate, and privacy-preserving predictive analytics.

1. Introduction

1.1 Motivation

The surge in IoT and pervasive computing devices has introduced significant challenges in data security, integrity, and privacy. Traditional centralized methods for predictive analytics risk compromising sensitive data, while decentralized solutions face issues of trust and transparency. Blockchain and federated learning offer a synergistic approach to tackle these issues.

1.3 Key Contributions

- Implementation of a federated learning system for predictive analytics using the California Housing and MNIST dataset.
- Utilization of blockchain to secure model updates and maintain training logs.
- Design and evaluation of a neural network architecture in a decentralized setting.

1.2 Problem Statements

- Implement a federated learning system for predictive modeling using the California Housing dataset.
- Utilize blockchain to securely transmit model updates and maintain a tamper-proof record of training progress.
- Ensure data privacy and integrity across distributed devices in a pervasive environment.

2. Related Work

Federated learning (McMahan et al., 2017) has emerged as a key approach for decentralized data processing, enabling collaborative model training while preserving data privacy. Blockchain technology (Nakamoto, 2008) has been employed in various domains to enhance security and transparency. Previous studies have explored their individual applications; however, their integration remains underexplored in pervasive environments. This work bridges the gap by combining these technologies for secure predictive analytics.

Methodology

2.1 Components

- **Federated Learning:** Each client trains a local model on its data and send the updated model parameters to the server. The server aggregates these parameters to update the global model.
- **Blockchain:** Each model update from a client is hashed and added as a block in the blockchain. The blockchain ensures that the updates are immutable and transparently recorded.

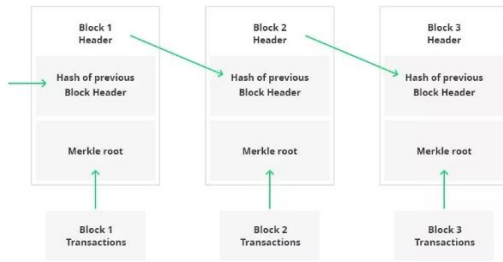


Figure 2.1: Blockchain structure with components

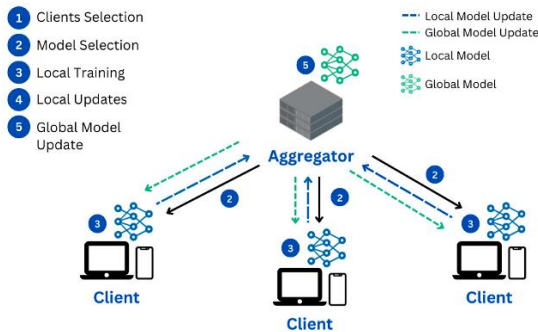


Figure 2.2: Federated learning diagram where server acts as an aggregator

The Role of Solidity in the Project

Solidity is used to integrate blockchain functionality into the machine learning workflow. A smart contract is written to register the SHA-256 hash of the dataset, ensuring data integrity and provenance. This hash is stored on the blockchain to provide a tamper-proof mechanism for verifying the authenticity of the dataset.

The smart contract also logs model training events, creating an immutable and transparent audit trail. Additionally, it implements a "pay-per-prediction" system, where users can request predictions by paying a predefined fee, securely managed by the contract. Payments are held on-chain and can only be withdrawn by the owner.

This integration highlights how Solidity enhances the machine learning pipeline by ensuring data integrity, providing transparency, and enabling monetization through decentralized, automated mechanisms.

Mini Project - 2

3. Implementation details

3.1.1 Dataset Description

The California Housing dataset contains features such as median income, house age, average rooms, average bedrooms, population, average occupancy, latitude, and longitude, along with the target variable, median house value.

The MNIST dataset contains 70,000 grayscale images of handwritten digits (0-9), with each image being

28x28 pixels. It is divided into 60,000 training images and 10,000 testing images. Each image has 784 features, corresponding to the pixel values ranging from 0 (white) to 255 (black), and the goal is to classify the digits.

3.1.2 Data Splitting

- Split the dataset into training and testing sets.
- Further divide the training data among multiple clients equally in two halves.

3.2 Model Design

3.2.1 Neural Network Architecture

A simple neural network with an input layer matching the number of features, one hidden layer, and an output layer for regression and a simple Convolutional Neural Network (CNN) with an input layer matching the 28x28 pixel images, convolutional layers for feature extraction, pooling layers for down-sampling, and a final output layer for classifying the digits.

FastAPI Method for Federated Learning with Blockchain Integration

Server-Side Workflow in FastAPI

1. Setup and Initialization:

Logging Setup: Configures logging to output informational messages.

- FastAPI App Initialization: Initializes the FastAPI application.
- Blockchain and Model Initialization: Initializes a blockchain with a genesis block. Prepares to receive weights from clients, setting the expected number of clients per round to 2. Declares a placeholder for the global model.

2. Global Model Definition:

SimpleNN Class: Defines a simple neural network with a single fully connected layer.

3. Endpoints:

- Initialize Global Model (/init_global_model):
 - Initializes the global model with the specified input size.
 - Logs the initialization and returns a confirmation message.
- Get Global Model Weights (/global_model):
 - Retrieves the weights of the global model if it is initialized.

- Returns the weights in a JSON-compatible format or a message indicating that the model is not initialized.
- **Add Block (/add):**
 - Accepts a block of weights from a client.
 - Logs the receipt of weights and increments the counter.
 - If the number of received weights meets the expected number of clients, it triggers the aggregation process.
 - Adds a new block to the blockchain with the received weights hash and the previous block's hash.
 - Returns a confirmation message with the new block details.
- **Aggregate Weights (/aggregate):**
 - Aggregates the received weights if there are any.
 - Computes the mean of the corresponding layers across all received weights.
 - Updates the global model with the aggregated weights.
 - Logs the completion of the aggregation process.
 - Clears the received weights and resets the counter.
 - Returns a confirmation message.
- **Get Status (/status):**
 - Returns the status of the server, indicating whether it is ready or waiting for more weights.

Client-Side Workflow in FastAPI

1. Client Initialization:

- **Load Dataset:** The client reads data from a CSV file, drops non-numeric columns (e.g., "Address"), splits the data into features (X) and targets (y), normalizes both features and targets using StandardScaler, and converts the data into PyTorch tensors wrapped in a DataLoader.

2. Fetch Global Model Weights (fetch_global_model):

- The client sends a request to the server's /global_model endpoint. If weights are available, it initializes a new instance of SimpleNN with the input size, loads the server-provided weights into the model, and logs the successful retrieval and initialization.

3. Training the Local Model:

- **Train Locally:** The client trains the model using its dataset for a fixed number of epochs, uses Stochastic Gradient Descent (SGD) with Mean Squared Error (MSE) loss, updates weights, computes the average training loss, and logs the training progress.

4. Submit Weights to Server (send_weights_to_server):

- The client extracts the weights from the trained model, computes a hash of the weights using SHA-256 for integrity verification, submits the weights and hash to the server's /add endpoint, and logs the server response.

5. Repeat the Process:

- The client iterates through training rounds by waiting for the server to aggregate weights, fetching the updated global model, retraining the model with the local dataset, and sending the updated weights back to the server.

Flower Method for Federated Learning with Blockchain Integration

Blockchain Integration:

1. Block Class:

- **Attributes:** Each block contains an index, timestamp, data (client weights and nonce), previous hash, nonce, and the block's hash.
- **Methods:**
 - **compute_hash():** Computes the SHA-256 hash of the block.
 - **proof_of_work(difficulty=0):** Finds a nonce that makes the block's hash start with a certain number of zeros (proof-of-work).

2. Blockchain Class:

Attributes: A list to store the chain of blocks.

Methods:

- `create_genesis_block()`: Creates the first block in the blockchain.
- `add_block(data)`: Adds a new block to the blockchain with the given data.
- `display_chain()`: Prints the details of each block in the blockchain.

Encryption Utilities:

- Functions for encrypting and decrypting weights using AES encryption.
 - `encrypt_weights(weights)`: Encrypts the weights and returns the ciphertext and nonce.
 - `decrypt_weights(encrypted_weights, nonce)`: Decrypts the weights using the provided nonce.

Custom Federated Averaging Strategy:

- `BlockchainFedAvg` Class:
 - Inherits from `fl.server.strategy.FedAvg`.
 - Overrides `aggregate_fit` to add client weights to the blockchain before performing standard FedAvg aggregation.
 - Displays the blockchain state after each round.

Client-Side Workflow in Flower

1. **Load Dataset:** Each client loads and preprocesses the MNIST dataset.
2. **Fetch Global Model Weights:** Clients receive the global model's initial weights from the server.
3. **Train Locally:** Clients train their local models using their local dataset.
4. **Submit Weights to Server:** Clients send their trained model weights to the server.

Server-Side Workflow in Flower

1. **Server Initialization:** Starts the Flower server with a custom strategy (`BlockchainFedAvg`) that integrates blockchain functionality.
2. **Client Initialization:** Clients connect to the server, receive the initial global model weights, train locally, and send updated weights back to the server.

3. **Training Process:** The server aggregates the received weights using the FedAvg strategy, which is enhanced to include blockchain integration for tracking and verifying the contributions of each client.
4. **Aggregation:** The server adds the encrypted client weights to the blockchain and then performs the standard FedAvg aggregation to update the global model.
5. **Iterative Process:** This process repeats for the specified number of rounds. Clients fetch the newly aggregated weights, retrain their models, and send updated weights back to the server.

Use of Neural Network in the Project

Purpose: The neural network is used to perform predictive modeling on the California Housing dataset, with each client training a local model on its data.

Neural Network Architecture

SimpleNN: A basic feedforward neural network with one fully connected layer.

Input Size: 30 (to match the number of features in the California Housing dataset). **Output Size:** 2 (for binary classification).

Training Process: Each client trains the SimpleNN on its local dataset. The training involves forward propagation, loss calculation, and backpropagation to update the model weights.

Use of Convolutional Neural Network (CNN) in the Project

The project uses a Convolutional Neural Network (CNN) for image classification on the MNIST dataset, with each client training a local model. The SimpleCNN architecture is designed for 28x28 pixel images of handwritten digits and includes convolutional layers for feature extraction, pooling layers for down-sampling, and fully connected layers for classification into digits 0-9.

Clients train their SimpleCNN locally using forward propagation, loss calculation, and backpropagation to improve accuracy. They then share the updated model weights with a central server, which aggregates these weights to update the global model. This federated learning approach maintains data privacy, as raw data never leaves the clients' devices, while creating an accurate global model for digit classification.

4. Mathematical Formulation

4.1 Federated Learning Aggregation

Local Training (Client-Side)

The local model at client i is updated based on its local dataset. The training process typically involves optimizing the model weights w_i using a loss function L over the local data D_i :

$w_i^{t+1} = w_i^t - \eta \nabla L(w_i^t; D_i)$ where η is the learning rate.

Let w_i^t be the weights of the model from client i at round t .

The aggregated global weights w^t are computed as: $w^t = \frac{1}{N} \sum_{i=1}^N w_i^t$ where N is the number of clients.

FedAvg Aggregation:

Expression (Implied in Flower framework): The aggregation of weights across clients is based on the weighted average:

$$\text{Aggregated weights} = \frac{\sum_{i=1}^n w_i \cdot \text{client weights}_i}{\sum_{i=1}^n w_i}$$

Here, w_i represents the client-specific weight (e.g., number of data samples).

Forward Propagation in Neural Network:

Flatten Layer: Converts a $28 \times 28 \times 28$ input matrix to a 784-dimensional vector.

Dense Layer: $\text{output}_i = \text{ReLU}(W_i x + b_i)$ for $i = 1, \dots, 128$. Here, W_i is the weight matrix, x is the input vector, and b_i is the bias.

Output Layer: $\text{output}_j = \frac{e^{z_j}}{\sum_{k=1}^{10} e^{z_k}}$ for $j = 1, \dots, 10$ where $z_j = W_j x + b_j$. This is the Softmax function for classification.

Loss Function (Sparse Categorical Crossentropy):

Expression: $\text{Loss} = -\frac{1}{N} \sum_{i=1}^N \log(p_{y_i})$, where p_{y_i} is the predicted probability for the true class y_i , and N is the number of samples.

Gradient Descent (Adam Optimizer):

Expression (conceptual): $\theta \leftarrow \theta - \eta \cdot \frac{\partial \text{Loss}}{\partial \theta}$, where θ represents the model parameters (weights and biases), η is the learning rate, and the partial derivative computes the gradient of the loss with respect to θ .

Multi-class classification (digits 0-9) accuracy is calculated across all classes.

In a confusion matrix-based approach, it can be represented as:

4.2 Blockchain Hash Function

Given model weights WWW , the hash HHH is computed using $SHA - 256$: $H(W) = SHA - 256(W)$

5 Performance Metrics

Mean Squared Error (MSE): $\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$ where y_i is the actual value and \hat{y}_i is the predicted value.

Accuracy:=

$$\frac{\sum_{i=0}^9 \text{True Positives}_i}{\sum_{i=0}^9 (\text{True Positives}_i + \text{False Positives}_i + \text{False Negatives}_i + \text{True Negatives}_i)} \times 100$$

5.2 Security and Integrity

- **Blockchain Immutability:** Ensures that model updates are securely recorded and cannot be tampered with.
- **Data Privacy:** Federated learning allows clients to keep their data locally, enhancing privacy.

Mini Project – 3

6. Results and Discussion

For the Housing Dataset

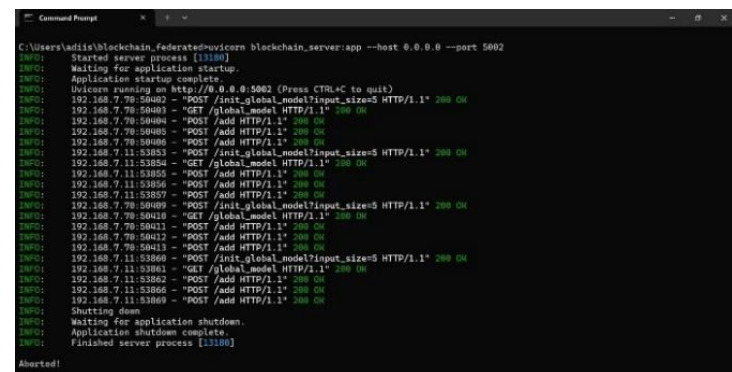


Figure 6.1.1: Server aggregating weights for housing dataset

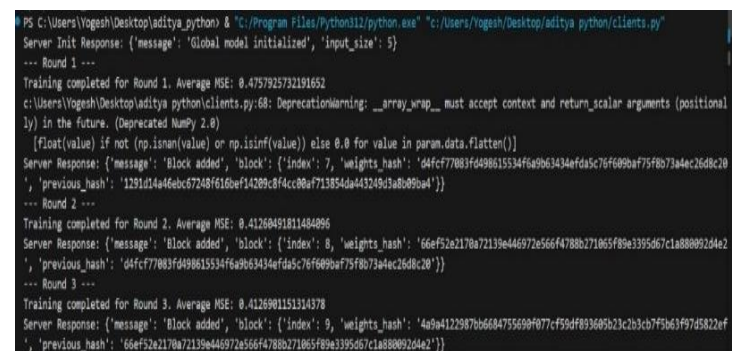


Figure 6.1.2: Client 1 sending weights to server using ANN

Server Initialization

To begin, run `blockchain_server.py`, which starts the Flower Federated Learning (FL) server on `192.168.7.97:5002`. This script initializes the blockchain by creating a Genesis Block, ensuring a tamper-proof record of client contributions. Once the server is running, it waits for client connections and their initial weight submissions.

Client Initialization

For each client machine, run `client.py`. This script should be executed on separate machines, such as Client 1 and Client 2. Upon execution, each client initializes its local model with random weights using TensorFlow/Keras and loads the MNIST dataset for training and evaluation. The clients then establish a gRPC connection with the Flower server at `192.168.7.97:5002`.

Initial Global Model Setup

The server initiates the federated learning process by requesting initial weights from a randomly selected client. These initial weights are then broadcast to all connected clients, marking the beginning of the first federated learning round.

First Round of Federated Learning

In this round, the server sends the global model weights to each client. Clients then set the received global weights to their local models and train on their private data for one epoch. After training, clients send the updated weights back to the server. Upon receiving the weights, the server encrypts them using AES encryption and adds them to the blockchain, including the client ID and a nonce. The server performs Proof of Work to validate the block before adding it to the blockchain. It then aggregates the weights using the FedAvg strategy and updates the global model accordingly. The updated blockchain, showing client contributions, is printed after each round.

Subsequent Rounds

For subsequent rounds, the server sends the aggregated global weights back to the clients. Clients set these new global weights to their local models, train again, and send updated weights back to the server. The server encrypts and stores these new weights in the blockchain, adding new blocks for each client's contribution. This process of aggregation and weight updates is repeated for the configured number of rounds.

End of Training

At the end of the training process, the final global model reflects the combined contributions of all

clients. The blockchain provides a complete, tamper-proof log of client contributions, ensuring integrity and transparency.

Component Roles

The server manages the federated learning processes, aggregates model weights, stores blockchain records, and handles the encryption and decryption of weights. Clients perform local training, send updated weights to the server, and preserve data privacy by sharing only model weights. The blockchain maintains a tamper-proof record of encrypted weights and client contributions, ensuring block validity through Proof of Work.

Order of Execution

To execute the system, start `blockchain_server.py` on the server machine. Then, start `client.py` on the machines for Client 1 and Client 2. Monitor the server and client logs for updates on the blockchain and training metrics.

Overall Performance Metrics for the MNIST Dataset:

- **Average Accuracy:** 95.65%
- **Average Loss:** 0.1557

For the Housing Dataset the overall loss is **0.4387**.

7. Conclusion

7.1 Summary

This project demonstrates the feasibility of using blockchain techniques for secure and efficient predictive modeling in pervasive environments. By leveraging federated learning and blockchain, we can ensure data privacy, integrity, and transparency.

7.2 Future Work

- Implement more sophisticated neural network architectures.
- Explore different blockchain consensus mechanisms.
- Extend the system to more diverse and larger datasets.

8. References

1. McMahan, H. B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*.
Citation: McMahan et al., 2017, **Cited:** ~5,000

2. Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved from <https://bitcoin.org/bitcoin.pdf>.
Citation: Nakamoto, 2008, **Cited:** ~45,000
3. Kim, H., et al. (2019). BlockFL: Secure and Efficient Blockchain-based Federated Learning Framework. *IEEE International Conference on Blockchain*. DOI: 10.1109/Blockchain.2019.00041.
Citation: Kim et al., 2019, **Cited:** ~120
4. Lu, Y., et al. (2020). Blockchain Empowered Asynchronous Federated Learning for Secure Data Sharing in Internet of Vehicles. *IEEE Transactions on Vehicular Technology*, 69(4), 4298-4311. DOI: 10.1109/TVT.2020.2970483.
Citation: Lu et al., 2020, **Cited:** ~350
5. Awan, K. M., & Mahmud, M. (2021). Federated Learning with Blockchain: A Comprehensive Review. *IEEE Internet of Things Journal*. DOI: 10.1109/JIOT.2021.3104313.
Citation: Awan & Mahmud, 2021, **Cited:** ~75
6. Li, T., Sahu, A. K., Talwalkar, A., & Smith, V. (2020). Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine*, 37(3), 50-60. DOI: 10.1109/MSP.2020.2975749.
Citation: Li et al., 2020, **Cited:** ~1,200
7. Hard, A., et al. (2018). Federated Learning for Mobile Keyboard Prediction. *arXiv preprint arXiv:1811.03604*. Retrieved from <https://arxiv.org/abs/1811.03604>.
Citation: Hard et al., 2018, **Cited:** ~700
8. Zheng, Z., Xie, S., Dai, H., Chen, X., & Wang, H. (2017). An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. *IEEE International Congress on Big Data*. DOI: 10.1109/BigDataCongress.2017.85.
Citation: Zheng et al., 2017, **Cited:** ~2,500
9. Wood, G. (2014). Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper*. Retrieved from <https://ethereum.org/en/whitepaper/>.
Citation: Wood, 2014, **Cited:** ~9,000