**Title**

Detection of Bank Transaction Anomalies using Gradient Boosted Federated Learning

**Journal**

IEEE Access

**Authors**

Rohan Chandrashekar

Rithvik Grandhi

Rahul Roshan G

Dr. Shyalaja SS

**Publication Status**

Submitted for Publication in November 2024

Peer reviewed

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

*Digital Object Identifier 10.1109/ACCESS.2023.1120000*

# Detection of Bank Transaction Anomalies using Gradient Boosted Federated Learning

**ROHAN CHANDRASHEKAR, RITHVIK GRANDHI, RAHUL ROSHAN G and SHYLAJA SS**
Department of Computer Science and Engineering, PES University, 100 Feet Ring Road, Banashankari 3rd Stage, Bengaluru 560085, India
Corresponding author: Rohan Chandrashekar (e-mail: chandrashekar.rohans@gmail.com).

**ABSTRACT** The advent of the internet has brought forth a new era in digital technology. This has revolutionized many businesses and sectors particularly banking and retail. Although this transformation has solved many problems and created numerous opportunities, it has also given rise to many challenges, the foremost being security. The increase in fraudulent transactions has been a major concern for both banks and customers. While customers are at risk of losing their savings, fraudulent transactions negatively affect the reputation of retail banks. The accurate and timely detection of bank transactions as fraudulent ranks among the most critical challenges confronting the banking sector. This study aims to research and introduce two novel machine learning techniques to address this pressing challenge. We leveraged the decentralized capabilities of the Federated Learning framework to bolster data security and privacy while analyzing sensitive and confidential data from banking transactions. What sets our methods apart is the strategic application of federated learning with gradient-boosting algorithms, celebrated for their adeptness in managing significantly imbalanced datasets. This innovative approach not only enhances our ability to detect anomalies in banking transactions but also minimizes false positives and achieves greater accuracy. The outcome of our research shows that by adopting these techniques, we can mitigate data privacy concerns and achieve significant improvements in both the detection of anomalies and the efficiency of the process. Thus, it paves the way for a transformative impact of this method in the realm of contemporary online banking security.

**INDEX TERMS** Anomaly detection, banking, client, data privacy, decentralized, federated learning, gradient boosting, machine learning, privacy, security, server, transactions.

## I. INTRODUCTION

THE banking and retail sectors have undergone a paradigm shift in the recent years. This shift included the adoption of online banking by both customers and businesses. While this has significantly improved ease of use, speed, efficiency, and customer satisfaction in online transactions, it has also introduced multiple layers with added complexity. This allows hackers and other unethical parties to exploit vulnerabilities that may exist within this architecture.

Although numerous security measures have been developed and implemented, this problem has persisted and continues to grow. It has been estimated that between 2% and 5% of the global Gross Domestic Product (GDP) is subjected to money laundering [1]. To avoid compromising the security structure and recovering funds, the early detection and reporting of illicit transactions are of paramount importance. The stakes are so high that, states have institutions and banks have divisions dedicated to Anti-Money Laundering (AML) activities [2] [3]. This is a pressing aspect of this problem. However, despite these measures, only a very small amount (approximately 1%) of the money laundered was seized.

These countermeasures provide only an intermediate solution for rapidly evolving and dynamic problems. Attackers are constantly improving their skills and developing sophisticated methods for intercepting and manipulating bank transactions. Hence, it is necessary to build our models such that they immediately identify security breaches and are capable of adapting to new and more advanced attack methods.

This gives rise to a critical issue of data security [4]. A conventional machine learning approach involves a central server that handles data storage and training models. However, this type of centralized learning involves the repeated transfer of data and hence is not viable.

We found that Federated Learning (FL) is a unique and robust method for addressing this issue. As shown in Figure 1, FL is built on the foundations of decentralized learning, and we can use this framework to provide our training models to clients [5]. The models were trained at the client level and

their updates were sent back to the server for aggregation. Subsequently, one consolidated model was transferred back to the clients using tenets of distributed computing. This would be advantageous for identifying real-time threats and providing visibility and training to other clients, even if they were not attacked.

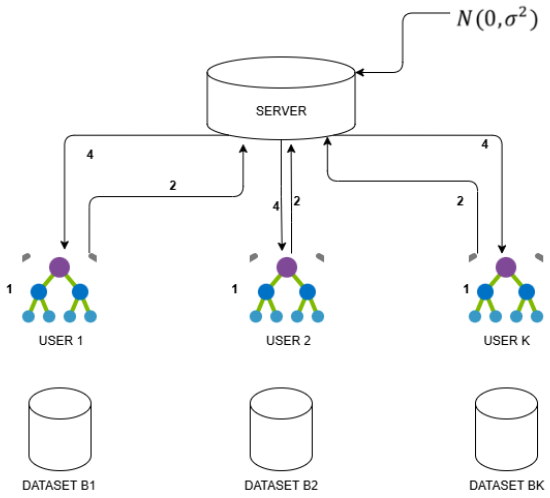


**FIGURE 1.** High-Level overview of Federated Learning

However, the decentralized nature of federated learning has mostly been used to handle deep learning models. However, as we will show, such deep learning models fall short because of the heterogeneity of the banking data.

Therefore, we decided to use gradient boosting as our machine learning training model to train clients to identify bank transaction anomalies [6]. These models are well-known for their ability to handle complex and nonlinear data. This unique and novel approach allows us to leverage the combined capabilities of distributed computing to preserve data security, as well as gradient boosting for anomaly detection of imbalanced data.

## II. BACKGROUND AND RELATED WORK

In recent years, anomaly detection in financial transactions has undergone significant advancements, particularly with the integration of machine learning and federated learning techniques. This section reviews related works that have laid the foundation for the current study, highlighting key contributions, methodologies, and findings in the literature.

### A. TRADITIONAL MACHINE LEARNING METHODS

#### 1) Statistical Methods

Early approaches to anomaly detection in financial transactions relied heavily on statistical methods. Techniques such as the Z-score, regression models, and clustering were used to identify outliers in the transaction data. These methods, although straightforward and interpretable, often struggle with high-dimensional data and complex fraudulent patterns.

Bolton and Hand [7] highlight the limitations of statistical methods in handling evolving fraud tactics.

#### 2) Rule Based Systems

Rule-based systems are another prevalent approach, in which domain experts define specific rules and thresholds to flag suspicious transactions. These systems provide high interpretability but are limited by their rigidity and inability to adapt to new fraud patterns.

Gopal et al. [8] developed a rule-based system for anomaly detection in IP flow. However, the static nature of such systems requires frequent updates and adjustments.

### B. MACHINE LEARNING APPROACHES

#### 1) Supervised Learning

Supervised learning models introduce more sophisticated approaches to anomaly detection.

Zhang et al. [9] utilized decision trees and random forests to detect financial fraud, demonstrating significant improvements over the traditional methods.

Support Vector Machines (SVMs) have also been used for anomaly detection by leveraging their ability to handle high-dimensional spaces. [10]

#### 2) Unsupervised Learning

Given the scarcity of labeled fraud data, unsupervised learning techniques such as clustering and autoencoders have gained popularity. These methods identify anomalies by detecting deviations from established patterns in data.

Techniques such as k-means clustering and DBSCAN have been used to cluster the data and identify outliers. These approaches highlight the potential of limited labeled data [11].

Deep learning-based autoencoders were utilized to reconstruct transaction data and flag anomalies based on reconstruction errors. A notable example is the work of Zamini et al. [12], who applied autoencoders to detect network intrusions and financial fraud and demonstrated their effectiveness in capturing complex patterns.

### C. FEDERATED LEARNING FRAMEWORK

Federated learning has emerged as a powerful framework to address privacy concerns in collaborative model training. The federated averaging method introduced by McMahan et al. [13] aggregates the model updates from multiple clients to form a global model. This approach laid the groundwork for federated learning studies.

Recent studies have applied federated learning to anomaly detection in financial transactions, focusing on privacy preserving model training. Mothukuri et al. [14] proposed a federated anomaly-detection framework for IoT data. Their approach demonstrated the feasibility of federated learning for real-time anomaly detection.

Yang et al. [15] developed a federated learning framework for fraud detection in mobile payment systems, achieving a

high detection accuracy while ensuring data privacy. Their study highlighted the scalability and effectiveness of federated learning in detecting financial fraud.

## III. THE PROBLEM STATEMENT

Financial institutions are constantly challenged by the pervasive issue of fraudulent transaction. This results in huge financial losses and erosion of customer trust [16]. Thus, detecting anomalous fraudulent transactions is critical. However, this is fraught with numerous technical challenges. Traditional machine learning models have been conventionally employed for anomaly detection. However, these methods have several limitations [17].

### A. DATA PRIVACY AND SECURITY CONCERNS

Financial data are very sensitive, and their centralized collection poses several significant risk factors. Different regulatory requirements and privacy laws such as GPDR and CCPA impose stringent constraints on the storage and processing of such data [18]. Thus, rendering centralized approaches untenable.

### B. HETEROGENEITY OF DATA

Bank transaction data are inherently heterogeneous and originate from various sources, with different collection formats, structures, and transaction patterns. This complicates the centralized aggregation and processing of data, leading to suboptimal model performance.

### C. IMBALANCED DATA DISTRIBUTION

Fraudulent transactions usually represent a very small portion of the total transaction volume, creating a highly imbalanced dataset that leads to a high rate of false negatives (i.e., fraudulent transactions classified as legitimate), which can be particularly damaging in a financial context.

### D. SCALABILITY ISSUES

As transaction volume increases, centralized models face significant scalability challenges. The computational and storage demands for processing and analyzing vast amounts of data in a centralized manner can be prohibitive [19].

## IV. PROPOSED SOLUTION

To address these challenges, this study proposes the use of gradient-boosted federated learning, which is a decentralized approach to machine learning. We explore two methodologies within this framework.

### A. CYCLIC XGBOOST FEDERATED LEARNING

Here, federated learning is executed in a sequentially client-by-client. Only one client participated in each training round instead of aggregating the data from multiple clients. The XGBoost trees trained by one client were passed to the next client to serve as the starting model for the next round of boosting.

### B. XGBOOST BOOTSTRAP AGGREGATION BASED FEDERATED LEARNING

In this technique, each client acts as a bootstrap by randomly selecting the data subsets. During each federated learning round, clients enhance several trees based on the local subsets. The trees from all clients were then aggregated on the server and appended to the global model from the previous round to form a new global model.

## V. DATASET

The PaySim Simulator Dataset is used in this study. This dataset was presented in a study conducted by the Blekinge Institute of Technology, detailed in the thesis "PaySim Financial Simulator" by Ahmed Elmir [20]. This dataset was specifically designed to mimic real-world money transactions, making it an ideal resource for research on financial fraud detection [21]. Figure 2 shows the internal operation of PaySim.

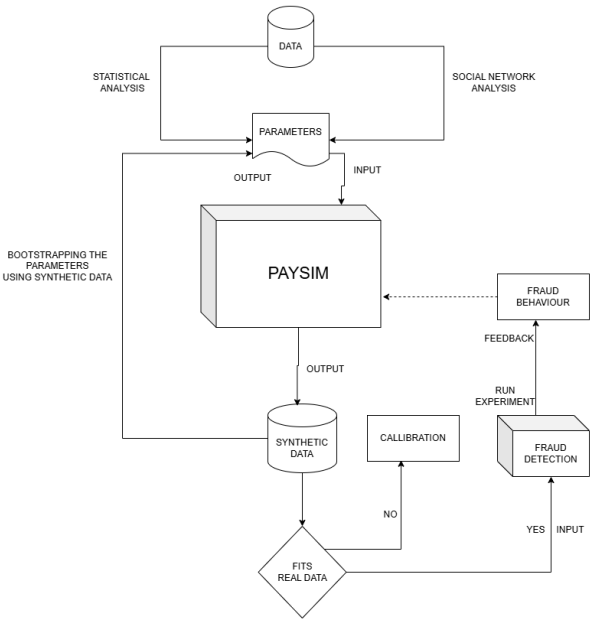


**FIGURE 2.** Internal working of the PaySim Data Simulator

### A. DATA COLLECTION

The dataset was generated using PaySim Financial simulator. This simulator was parameterized through the "paysim.properties" file, which allowed us to adjust various aspects of the simulation to match specific research requirements to the scenarios. The key parameters that were tweaked include the following.

#### 1) Transaction Rates

The rates of different transaction types (e.g., CASH-IN and CASH-OUT) reflect realistic transaction frequencies.

#### 2) Fraud Ratios

We set the proportion of fraudulent transactions to study the impact of different fraud rates on detection models.

#### 3) Account Behaviour

Modifying parameters related to account creation, transaction limits, and account balances to simulate different user behaviours.

### B. DATASET DESCRIPTION

This method provides a rich and detailed dataset containing over six million records, effectively mirroring the complexities of banking activities, including legitimate and fraudulent transactions. The attributes of the dataset are presented in Table 1.

**TABLE 1. Raw Dataset Attributes**

| Attribute | Description |
|---|---|
| step | Maps a unit of time in the real world. In this case 1 step is 1 hour of time. |
| type | CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER. |
| amount | Amount of the transaction in local currency |
| nameOrig | Customer ID of the customer who has initiated the transaction |
| oldbalanceOrig | Customer's balance before the transaction |
| newbalanceOrig | Customer's balance after the transaction |
| nameDest | Merchant ID of the recipient of the transaction |
| oldbalanceDest | Merchant's balance before the transaction |
| newbalanceDest | Merchant's balance after the transaction |
| isFraud | Binary classification of a fraudulent transaction (1) and non-fraudulent transaction (0). |

### C. DATASET IMBALANCE

Clearly, in Figure 3, we can see that the values of the isFraud column are highly imbalanced, with a skewed majority of transactions labeled as not fraudulent (or 0). This reflects real-world scenarios.

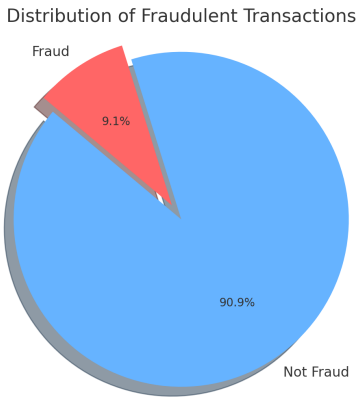


**FIGURE 3. isFraud Column Binary Distribution**

### D. PREPROCESSING

To prepare this dataset for effective XGBoosted Federated training, several cleaning and preprocessing steps were required. These are listed below.

#### 1) Cleanup

Identifying any incomplete records and effectively removing them.

#### 2) Removing Redundant Features

Identifying and eliminating features that do not contribute to the predictive training of the model and introduce noise. Here, we identify that *step*, *oldbalanceOrig*, *oldbalanceDest*, *newbalanceOrig*, and *newbalanceDest* do not contribute to the model's training for fraud detection, and hence are removed.

#### 3) One-Hot Encoding

Conversion of categorical features, such as transaction *type*, to numerical values using one-hot encoding. This leads to an increase in the dimensionality of the data, but does not adversely affect the model training performance.

#### 4) Tokenisation

Attributes such as *nameOrig* and *nameDest*, which include the customer and merchant id in alphanumeric values, are converted into numerical values using tokenization.

Table 2 presents the attributes of the processed data.

**TABLE 2. Processed Dataset Attributes**

| Attribute | Description |
|---|---|
| amount | Total value of transaction in local currency. |
| nameOrig | Tokenised value of the customer id. |
| nameDest | Tokenised value of the merchant id. |
| $CASH-IN$ | Binary value indicating if the transaction leads to cash inflow. |
| $CASH-OUT$ | Binary value indicating if the transaction leads to cash outflow. |
| *DEBIT* | Binary value indicating if it is a debit card transaction. |
| *PAYMENT* | Binary value indicating if the transaction is a payment to a merchant. |
| *TRANSFER* | Binary value indicating if the transaction is a payment to a customer. |
| newbalanceDest | Merchant's balance after the transaction |
| isFraud | Binary classification of a fraudulent transaction (1) and non-fraudulent transaction (0). |

## VI. FEDERATED LEARNING

Federated Learning (FL) is a new and innovative approach for training Machine Learning (ML) models [22]. FL enables the training of models across decentralized data sources, which allows the preservation of data privacy and security. Unlike conventional centralized ML, where all data are aggregated into a single repository for training, FL allows individual data owners (clients) to train models locally and only shares model updates with a central aggregator (server). Figure 4 shows the distributions of the client and server. This approach also reduces latency and leverages distributed computational resources [23] [24].

### A. KEY COMPONENTS OF FEDERATED LEARNING

#### 1) Clients

Clients in federated learning are decentralized entities, each holding a private dataset. These clients perform local training on the shared global model received from the central server. The local training process is represented by the following equation:

$$\theta_k^{(t+1)} = \theta^{(t)} - \eta \nabla L_k(\theta^{(t)}) \tag{1}$$

where $\theta^{(t)}$ is the global model at iteration $t$, $\eta$ is the learning rate, $L_k$ is the local loss function for client $k$, and $\theta_k^{(t+1)}$ represents the locally updated model parameters. After training, clients send only the updated parameters to the central server, ensuring that the raw data remain local and private.

#### 2) Server

The central server coordinates the federated learning process by aggregating the model updates from all clients. The aggregation step typically uses the Federated Averaging (FedAvg) algorithm [25] represented as

$$\theta^{(t+1)} = \frac{1}{K} \sum_{k=1}^{K} \theta_k^{(t+1)} \tag{2}$$

where $K$ is the number of participating clients, $\theta_k^{(t+1)}$ is the local model update from client $k$, and $\theta^{(t+1)}$ is the updated global model. The server then redistributes this updated global model to clients for the next round of local training, continuing the iterative process until the model converges.

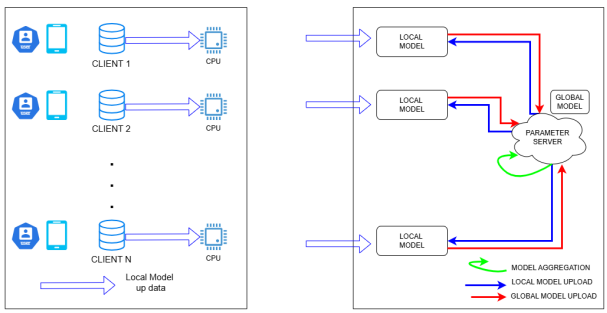


**FIGURE 4.** Client and Server distribution in FL

### B. WORKING OF FEDERATED LEARNING

#### 1) Initialization

The central server initializes the global model parameters $\theta^{(0)}$. These parameters can be initialized randomly or based on a pre-trained model. The initial global model $\theta^{(0)}$ is then distributed to all participating clients $k = 1, 2, \ldots, K$.

#### 2) Distribution

The central server sends the initial global model $\theta^{(0)}$ to all clients. Each client $k$ receives a copy of the global model denoted by $\theta_k^{(0)} = \theta^{(0)}$.

#### 3) Local Training

Each client $k$ holds its local dataset $D_k = \{(x_i^k, y_i^k)\}_{i=1}^{n_k}$, where $n_k$ is the number of data points on client $k$. The local training process involves minimizing the local loss function $L_k(\theta)$ using the dataset $D_k$.

The local model parameters were updated using a gradient based optimization algorithm. For a simple gradient descent update, the local model parameters $\theta_k$ are updated as follows:

$$\theta_k^{(t+1)} = \theta_k^{(t)} - \eta \nabla L_k(\theta_k^{(t)}) \tag{3}$$

where $\eta$ is the learning rate, and $\nabla L_k(\theta_k^{(t)})$ is the gradient of the local loss function with respect to the model parameters at iteration $t$.

This step is repeated for $E$ epochs or iterations to sufficiently train the local model using the client's data.

#### 4) Uploading Updates

After local training, each client $k$ sends its updated model parameter $\theta_k^{(t+1)}$ to the central server. To ensure data privacy, only the model updates (not raw data) were transmitted.

#### 5) Aggregation

The central server aggregates the updates received from all the participating clients to form a new global model. A typical aggregation method is Federated Averaging (FedAvg). The FedAvg algorithm calculates the new global model parameter $\theta^{(t+1)}$ as the weighted average of the client updates:

$$\theta^{(t+1)} = \frac{1}{N} \sum_{k=1}^{K} n_k \theta_k^{(t+1)} \tag{4}$$

where $n_k$ is the number of data points on client $k$, and $N = \sum_{k=1}^{K} n_k$ is the total number of data points across all clients. This weighted averaging ensures that updates from clients with more data have greater influence on the global model.

#### 6) Redistribution

The updated global model $\theta^{(t+1)}$ is redistributed to the clients by the central server. Each client $k$ receives the updated global model parameter $\theta_k^{(t+1)} = \theta^{(t+1)}$.

#### 7) Iteration

Steps 3-6 are repeated for a predetermined number of communication rounds or until the global model parameters converge. The iterative process ensures continuous improvement of the model. Mathematically, the convergence criteria can be defined as

$$\|\theta^{(t+1)} - \theta^{(t)}\| < \epsilon \tag{5}$$

where $\epsilon$ is a small positive threshold representing the convergence tolerance.

#### 8) Convergence and Deployment

The iterative process continues until the global model converges to an optimal solution. Upon convergence, the final global model $\theta^{(T)}$ (after $T$ rounds) is deployed. Optionally,

each client can use the final global model locally for specific applications or predictions.

## VII. DRAWBACKS OF CONVENTIONAL DEEP LEARNING MODELS

Although conventional supervised deep learning algorithms have proven to be very effective in solving most real-world classification problems, they have been shown to underperform anomaly detection tasks. Here, we list some of the drawbacks encountered when considering a supervised deep learning strategy for our training model.

### A. DATA PRIVACY AND SECURITY CONCERNS

Financial fraudulent transaction detection involves dealing with sensitive and confidential data. Deep learning models require centralized data collection for training purposes. These centralized data repositories and pipelines are vulnerable to exposure and raise privacy concerns [26]. Such models also struggle to comply with the strict laws of the GPDA and the CCPA.

### B. HIGH COMPUTATIONAL AND RESOURCE REQUIREMENTS

Deep learning models, specifically supervised machine learning algorithms, are known to have complex architectures and practical computational limits [27]. Deep learning networks, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), require significant computational resources and large amounts of data. This need can be a limiting factor in cost-constrained banking.

### C. ANOMALY DETECTION IN IMBALANCED DATASET

The high complexity of deep learning models with multiple layers and parameters makes it difficult to detect anomalies in data. These models excel at detecting patterns on which they have been trained but struggle to identify new anomalies that do not fit a set pattern. Conventional algorithms, such as gradient descent, are not optimized to handle sparse distributions; they prioritize minimizing a global loss function that can obscure the nuances of anomalies in data. These models are ineffective in detecting outliers and are better suited for pattern recognition [28] [29].

### D. COMPLEXITY AND EASE OF USE

Deep learning models often suffer from a lack of interpretability. Despite the availability of a large number of libraries and frameworks, their architecture is often presented as a black box. This lack of transparency is critical in sensitive environments such as fraud detection, where all stakeholders must be able to endorse the trust and compliance of the model. The developers need a very strong understanding of neural networks to build, deploy and monitor these models. This steep learning curve poses a significant challenge to the entire industry.

The mathematical foundations of deep learning algorithms show that their optimization processes are inherently biased towards the majority class in imbalanced datasets. This results in poor anomaly detection, because the models are less sensitive to rare atypical instances. Techniques such as modified loss functions, data augmentation, and specialized architectures are required to mitigate these issues. However, deep learning models cannot achieve satisfactory performance in anomaly detection tasks [30].

## VIII. XGBOOST

XGBoost, short for eXtreme Gradient Boosting, is a powerful and scalable machine learning algorithm that has gained widespread popularity owing to its superior performance in a variety of predictive modelling tasks. Developed by Chen et al. [31], XGBoost is an implementation of gradient-boosted decision trees, designed for speed and performance. It has become the cornerstone of many winning solutions in machine learning competitions and is widely used in industry and academia owing to its robustness, efficiency, and flexibility [32].

### A. WORKING OF XGBOOST

It was specifically designed for speed and performance. The algorithm combines gradient boosting principles with system optimization and algorithmic enhancements. Here, we detail the operation of the XGBoost algorithm, with a focus on its mathematical underpinnings and customization for anomaly detection in highly imbalanced datasets.

**Algorithm 1** XGBoost for binary classification

**Require:** Dataset $\{(x_i, y_i)\}_{i=1}^{n}$, learning rate $\eta$, number of trees $T$

Initialize global model parameters $\theta^{(0)}$
Initialize predictions $\hat{y}_i^{(0)} = \frac{1}{1+e^{-\text{base\_score}}}$
**for** $t = 1$ to $T$ **do**
  Compute pseudo-residuals $r_i^{(t)} = \hat{y}_i^{(t-1)} - y_i$
  Compute second-order gradients $h_i^{(t)} = \hat{y}_i^{(t-1)}(1 - \hat{y}_i^{(t-1)})$
  Construct tree $f_t$:
  **for** each leaf node $j$ **do**
    Compute sums of gradients $G_j = \sum_{i \in \text{leaf } j} r_i^{(t)}$
    Compute sums of Hessians $H_j = \sum_{i \in \text{leaf } j} h_i^{(t)}$
    Compute weight of leaf $j$: $w_j = -\frac{G_j}{H_j + \lambda}$
    Compute gain:

$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma \tag{6}$$

    Prune tree if Gain $< \gamma$
  **end for**
  Update predictions $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta f_t(x_i)$
**end for**
Final prediction $\hat{y}_i = \frac{1}{1+e^{-\hat{y}_i^{(T)}}}$

### B. BROADER UMBRELLA OF DECISION TREES

XGBoost falls under the broader umbrella of decision-tree algorithms, a class of models used for both classification and regression tasks. A decision tree is a flowchart-like structure, where internal nodes represent feature splits, branches represent the outcome of the split, and leaf nodes represent the final prediction [33]. Although decision trees are easy to interpret and implement, they tend to suffer from a high variance and overfitting when used alone.

To overcome these limitations, ensemble methods such as bagging and boosting have been developed. Gradient boosting is an ensemble technique that builds models sequentially, where each new model attempts to correct the errors made by previous models. XGBoost is an advanced version of gradient boosting that incorporates several enhancements to improve both the computational efficiency and model performance.

### C. HANDLING IMBALANCED DATASETS WITH XGBOOST

XGBoost is particularly well suited for handling imbalanced datasets owing to its flexibility and the variety of techniques it incorporates to address class imbalances [34]. Among other decision tree algorithms, such as AdaBoost, XGBoost has been shown to outperforms the other algorithms for binary label classification [35]. In this section, we describe the mechanisms and mathematical foundations that make XGBoost effective for imbalanced datasets.

#### 1) Scale Pos Weight

One of the key parameters in XGBoost for handling imbalanced datasets is `scale_pos_weight`, which balances the impact of positive and negative classes. This parameter adjusts the gradient and Hessian computation to assign a greater importance to the minority class.

In a binary classification problem, let $y_i$ be the true label and $\hat{y}_i$ be the predicted probability. The gradients $g_i$ and Hessian $h_i$ for the logistic loss function were computed as follows:

$$g_i = \hat{y}_i - y_i \tag{7}$$

$$h_i = \hat{y}_i(1 - \hat{y}_i) \tag{8}$$

With the `scale_pos_weight` parameter, these calculations are adjusted to:

$$g_i = \begin{cases} \hat{y}_i - y_i & \text{if } y_i = 0 \\ \texttt{scale\_pos\_weight} \cdot (\hat{y}_i - y_i) & \text{if } y_i = 1 \end{cases} \tag{9}$$

$$h_i = \begin{cases} \hat{y}_i(1 - \hat{y}_i) & \text{if } y_i = 0 \\ \texttt{scale\_pos\_weight} \cdot \hat{y}_i(1 - \hat{y}_i) & \text{if } y_i = 1 \end{cases} \tag{10}$$

This reweighting ensures that the minority class (often the positive class in the binary classification) has a greater impact on the model training process.

#### 2) Early Stopping

XGBoost uses early stopping to prevent overfitting, particularly in the majority classes. By monitoring the performance on a validation set, XGBoost can halt training when the improvement plateaus. This ensured that the model did not overfit the majority class at the expense of the minority class. The mathematical criterion for early stopping is as follows.

$$\text{Stop if } \left|\text{validation\_metric}^{(t)} - \text{validation\_metric}^{(t-1)}\right| < \epsilon \tag{11}$$

where $\epsilon$ is a small threshold value.

## IX. FLOWER

The Flower Federated Learning Framework, commonly referred to as flower, is an open-source federated learning framework developed to address the increasing need for scalable, flexible, and secure federated learning solutions. Flower was created to accommodate the growing diversity of data sources and machine learning frameworks, making federated learning accessible to a broader range of applications and research fields. Unlike earlier federated learning frameworks that were tightly coupled with specific machine learning libraries, Flower was designed from the ground up to be framework-agnostic, maximizing flexibility, and interoperability [36]. Figure 5 shows the internal working of the flower framework.

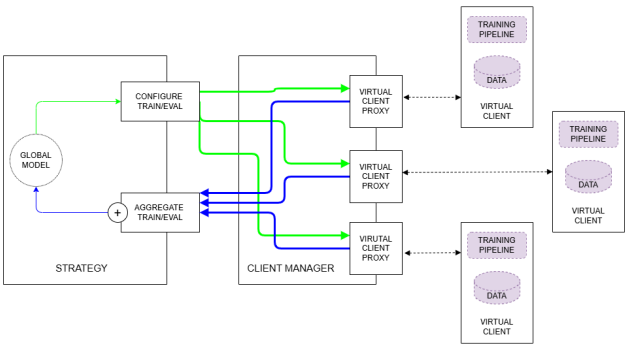


**FIGURE 5.** Flower Federated Learning Framework

### A. DISTINGUISHING FEATURES

Flower distinguishes itself from other federated learning frameworks through several key advantages:

#### 1) Machine Learning Agnostic

Flower's standout feature is its algorithm-agnostic nature. Unlike other federated learning frameworks that are often tied to specific machine learning libraries, Flower can be used with any machine learning framework that supports model serialization and deserialization. This includes, but is not limited to, TensorFlow, PyTorch, Scikit-learn, and XGBoost [37].

The algorithm-agnostic design is achieved through Flower's flexible client-server communication protocol. Clients can run any machine learning algorithm as long as they can serialize the model updates (e.g., weights and gradients) and send them back to the server. The server then aggregates the updates and redistributes the updated model. This approach allows flower to support a wide variety of machine learning tasks, from deep learning to ensemble methods, such as XGBoost.

#### 2) Scalability

Flower supports both cross-silo and cross-device federated learning, making it suitable for large-scale deployments. Its modular architecture and efficient communication protocols ensure that it can handle several clients with varying computational resources and network conditions.

#### 3) Customizability

Flower provides a high degree of customizability, allowing users to define federated learning algorithms, aggregation methods, and communication strategies. This makes it adaptable for a wide range of cases and experiments.

#### 4) Ease of Use

Flower offers a simple API that abstracts much of the complexity involved in setting up the federated learning experiments. This user-friendly interface has accelerated the development and deployment of federated learning models.

## X. FLOWER INTEGRATION WITH XGBOOST

Flower (FLWR) is a versatile federated learning (FL) framework designed to enable collaborative model training across multiple clients while preserving data privacy. XGBoost is a powerful and efficient implementation of gradient-boosted decision trees and is renowned for its performance in various machine learning tasks. Figure 6 shows federated averaging. By integrating with XGBoost, Flower allows efficient and privacy-preserving training of machine learning models, which is particularly beneficial for applications requiring stringent data privacy measures, such as detecting bank transaction anomalies [38].

### A. DATA PARTITIONING

Before the training began, the dataset was partitioned among the clients. Each client receives a unique subset of data for local training. Partitioning is performed in various ways, such as Independent and Identically Distributed (IID) or non-IID, depending on the desired data distribution across the clients.

#### 1) IID Partitioning

The data were evenly split such that each client obtained a similar distribution of samples.

#### 2) Non-IID Partitioning

The data are split to reflect real-world scenarios, where different clients may have different data distributions.

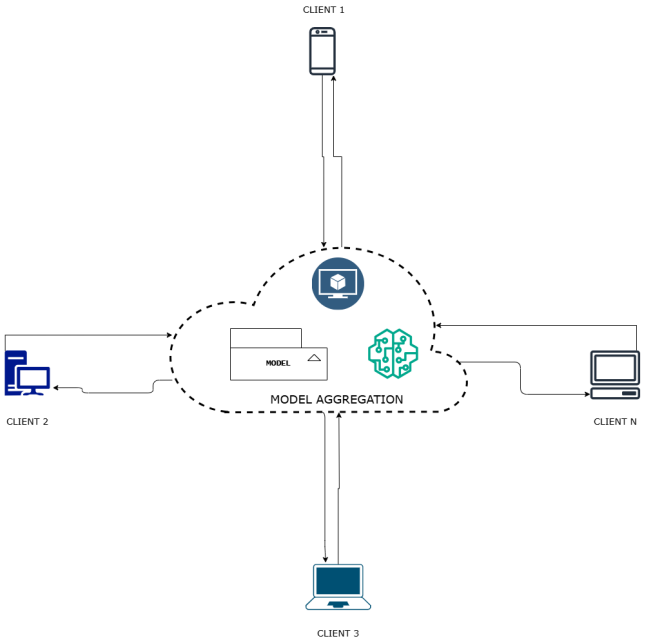


**FIGURE 6. XGBoost using Flower - Tree initialized from clients sent to server for aggregation**

### B. CLIENT-SIDE TRAINING

Each client independently trained the XGBoost model on its local dataset. The data were formatted into DMatrix format, which is the required input format for XGBoost. Local training involves building gradient-boosted decision trees based on the client data. The local models were updated through iterative training rounds, with each round comprising several boosting iterations. The training process ensures that the model captures the patterns and relationships inherent to the client's data.

### C. SERVER-SIDE AGGREGATION

The server plays a crucial role in aggregating models from the clients. Flower supports various aggregation strategies to combine models trained by individual clients into a global model:

#### 1) Cyclic Training

In this strategy, clients sequentially train the model. Only one client participated in each training round, passing the trained model to the next client for further training in the subsequent round. This cyclic approach ensures that the model is continuously refined, as it is exposed to data from different clients in each round. The cyclic nature of training helps to capture a wider variety of patterns and reduces the risk of overfitting to any single client's data.

#### 2) Bagging Aggregation

This strategy considers each client model as a bootstrap sample. In each federated learning round, all clients train a specified number of trees on their local data. These trees were then

sent to the central server where they were aggregated to form a new global model. Aggregation involves concatenating trees from all clients to enhance the stability and accuracy of the model. This approach leverages the diversity of client datasets to improve generalization.

## XI. DETECTION OF BANK TRANSACTION ANOMALIES USING GRADIENT BOOSTED FEDERATED LEARNING WITH CYCLIC TRAINING

Cyclic training in the flower federated learning framework involves iteratively updating a global XGBoost model by sequentially training it across multiple bank branches (clients) that retain their localized transaction data. Each branch trains the model based on its data and shares updates with the central server, which aggregates these updates and passes the improved model to the next branch in a cyclic manner. This approach ensures that diverse transaction patterns and anomalies are integrated into the global model, enhancing its generalization and accuracy in detecting fraudulent transactions, while preserving data privacy and security.

### A. DATA SPLITTING

The data splitting performed for cyclic training was the same as that performed for the bootstrap aggregation (bagging). Data were partitioned according to a predetermined strategy to ensure that the original data distribution was maintained as authentically as possible among the clients.

### B. FEDERATED LEARNING ENVIRONMENT SETUP

The flower-federated learning environment setup followed a procedure similar to that described above. Python scripts initiate and configure client and server setup. The clients are then configured to communicate securely with servers.

### C. ALGORITHMIC IMPLEMENTATION

The cyclic training process involved several detailed steps to ensure that each client sequentially contributed to refining the global model. This method helps systematically integrate diverse data distributions from different clients into a global model. The process is described as follows.

#### 1) Global Model Initialization

The server initializes the global model parameter $\theta_0$, setting up the XGBoost classifier. This initial model configuration includes setting hyperparameters, such as the learning rate, maximum depth of trees, and number of boosting rounds. These parameters serve as starting points for iterative refinement through contributions from multiple clients (bank branches). The server was responsible for orchestrating the training. It initializes the global XGBoost model and prepares to receive updates from clients during training rounds.

#### 2) Cyclic Training Process

**Client Selection and Local Training:** Clients are organized in a predetermined cyclic order. In each training round, a subset of clients is selected sequentially based on this order. The selected client performed the following tasks.

- **Data Loading and Preprocessing:** The client loads and preprocesses its local transaction dataset, ensuring that it is ready for training.
- **Local Model Training:**
  - **Training Configuration:** The client sets hyperparameters, such as the learning rate, maximum depth of trees, and number of boosting rounds.
  - **Training Loop:** The model underwent iterative training on local data. Each iteration involves computing the gradient of the loss function $L_k$ with respect to the model parameters, as follows $\theta_k^t$:

$$g_k^t = \nabla L_k(\theta_k^t) \tag{12}$$

The parameters are then updated using the computed gradients:

$$\theta_k^{t+1} = \theta_k^t - \eta g_k^t \tag{13}$$

Here, $g_k^t$ is the gradient at iteration $t$ for client $k$, and $\eta$ is the learning rate. This process iteratively refines the local model parameters.

After completing the local training iterations, client $k$ sends the updated parameter $\theta_k^T$ back to the server, where $T$ denotes the total number of local iterations.

**Global Model Aggregation:** Upon receiving updates from clients, the server aggregates these updates to form a new global model. This process involves:

- **Receiving Updates:** The server collects the updated model parameters from each participating client.
- **Parameter Aggregation:** The server combines the model parameters using a weighted averaging approach:

$$\theta^{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} \theta_k^{t+1} \tag{14}$$

Here, $n_k$ is the number of data samples at client $k$, and $n$ is the total number of samples across all clients. This weighted averaging ensured that clients with larger datasets had a more significant influence on the global model update.

**Iterative Refinement:** The cyclic training process is repeated for a predefined number of rounds or until a convergence criterion is met. Each round involved a new set of clients selected in cyclic order, ensuring that all clients contributed to the model training over time. Iterative refinement progressively improves the global model by incorporating knowledge from diverse data distributions present at different clients.

### D. RESULTS

We ran this model for up to 20 rounds of federated learning, measuring all performance metrics at each round and tracking the change in the values. We add clients incrementally to demonstrate performance improvement as the number of clients increases.

### 1) Time taken to run model

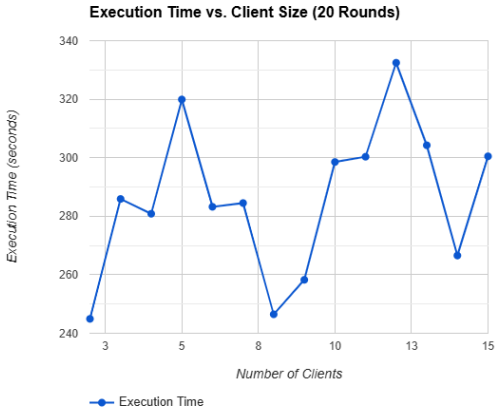


**FIGURE 7. Time taken to run the model v/s Number of clients for Cyclic XGBoost Federated Learning**

Figure 7 shows the amount of time required to run this model for 20 rounds with 2-16 clients. It started at approximately 240s for two clients and increased gradually as the number of clients increased.

It should be noted that this is only the time required to execute the model, and the amount of time required to prepare the dataset and format it in D-Matrix format as required by XGBoost, and subsequently splitting it among the clients, requires a significant amount of time as well.

### 2) Confusion Matrix

| No. of Clients | True Positives | False Positives | True Negatives | False Negatives |
|---|---|---|---|---|
| 2 | 957996 | 3345 | 9692 | 86004 |
| 3 | 957982 | 3359 | 9698 | 85998 |
| 4 | 957569 | 3772 | 9188 | 86508 |
| 5 | 957325 | 4016 | 8976 | 86720 |
| 6 | 957248 | 4093 | 9014 | 86682 |
| 7 | 957084 | 4257 | 9251 | 86445 |
| 8 | 957234 | 4107 | 9816 | 85880 |
| 9 | 957324 | 4017 | 9667 | 86029 |
| 10 | 957502 | 3839 | 9789 | 85907 |
| 11 | 957584 | 3757 | 10500 | 85196 |
| 12 | 957786 | 3555 | 11723 | 83973 |
| 13 | 957573 | 3768 | 11978 | 83718 |
| 14 | 958576 | 2765 | 15141 | 80555 |
| 15 | 958875 | 2466 | 15210 | 80486 |

**TABLE 3. Confusion Matrix Values for Different Number of Clients in Cyclic Configuration**

Table 3 presents the confusion matrices for different numbers of clients obtained after 20 rounds of training using Cyclic XGBoost Federated Learning.

### 3) Accuracy

Accuracy is a measure of the correctness of a machine learning model's predictions. It was calculated as the proportion of correct predictions out of the total number of predictions made. Mathematically, it is represented as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{15}$$

For binary classification, in which the outcomes are either positive or negative, the accuracy can be further broken down into true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Then, the formula becomes:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{16}$$

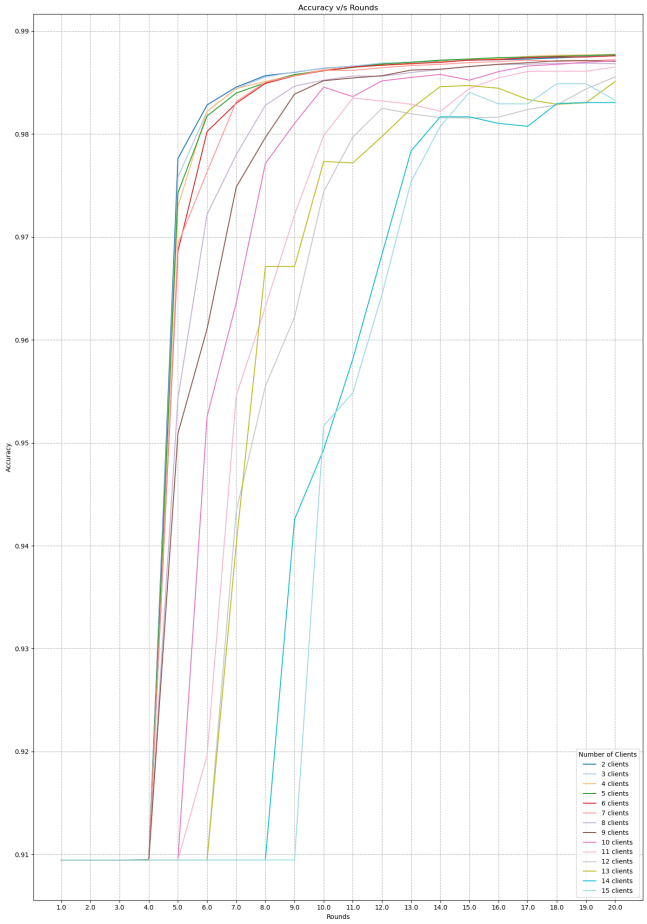


**FIGURE 8. Accuracy v/s No. of rounds for different clients using Cyclic XGBoost Federated learning**

As shown in Figure 8, there is a significant difference in how the accuracy changes before and after 13 rounds of federated learning. The initial accuracy of the model was 91% and we observe an improvement of 98% as more rounds of federated learning were processed. We can see a very steep improvement up to round 10 and a subsequent plateauing of the values for different clients. With respect to clients, there is a general trend toward better accuracy as the number of clients increases. However, as the number of clients increases beyond eight, the accuracy decreases slightly. This is because of the limited dataset available at our disposal. As this limited data is distributed among a larger number of clients, we can observe that each client will end up with less data to train on. Hence, we have reported the resulting accuracy values.

#### 4) Precision

Precision is a measure of the accuracy of positive predictions made using a machine learning model. This indicates the proportion of true positive predictions among all positive predictions (both true and false positives).

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}} \tag{17}$$

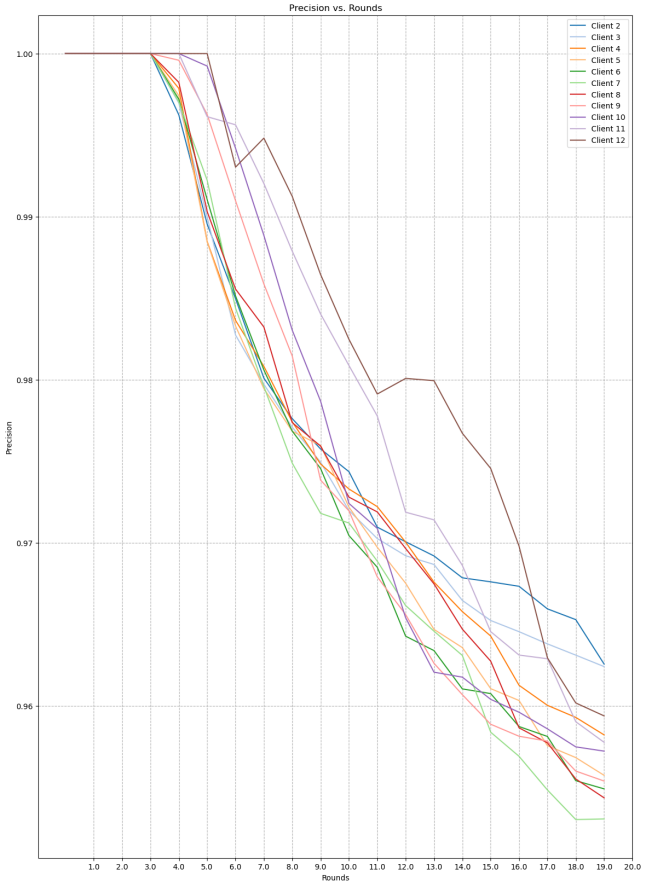


**FIGURE 9.** Precision v/s No. of rounds for different clients using Cyclic XGBoost Federated learning

As shown in Figure 9, precision tends to decrease as the number of rounds increases. This decline was more apparent in initial rounds (1-13) than in the later rounds (14-20). As the number of clients increased beyond 10, precision tended to decrease rapidly before stabilizing. This is in agreement with the results obtained for accuracy and can be explained by the limited size of the dataset.

#### 5) Recall

Recall, also known as the sensitivity or true positive rate, is a measure of how well a machine learning model identifies all relevant positive instances in a dataset. This indicates the proportion of true positive predictions among the total actual positive instances.

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}} \tag{18}$$

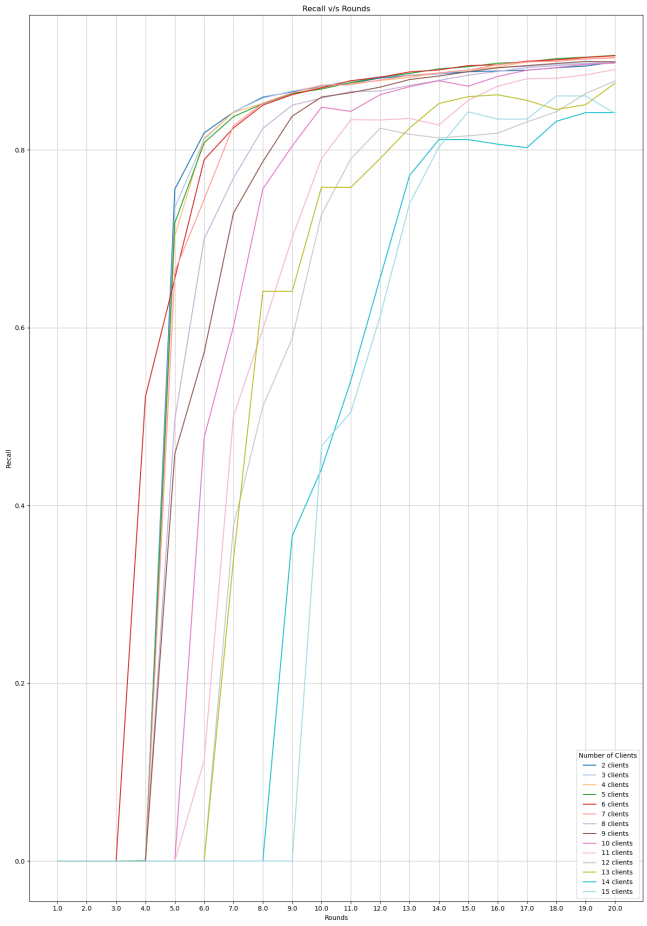


**FIGURE 10.** Recall v/s No. of rounds for different clients using Cyclic XGBoost Federated learning

In both representations shown in Figure 10, the recall value tended to improve as the number of rounds increased. This improvement was more rapid in the initial rounds (1-13) than in the later rounds (14-20). As the number of clients increased, we observed that the recall improved at a much faster rate, especially for more than 10 clients. However, as the number of rounds increased, the recall improvement tended to diminish, irrespective of the number of clients.

#### 6) F1 Score

The F1 Score is a measure of a model's accuracy that considers both precision and recall. This is the harmonic mean of the precision and recall, providing a single metric that balances the trade-off between them. The F1 Score is particularly useful when one needs to find a balance between precision and recall and there is an uneven class distribution.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{19}$$

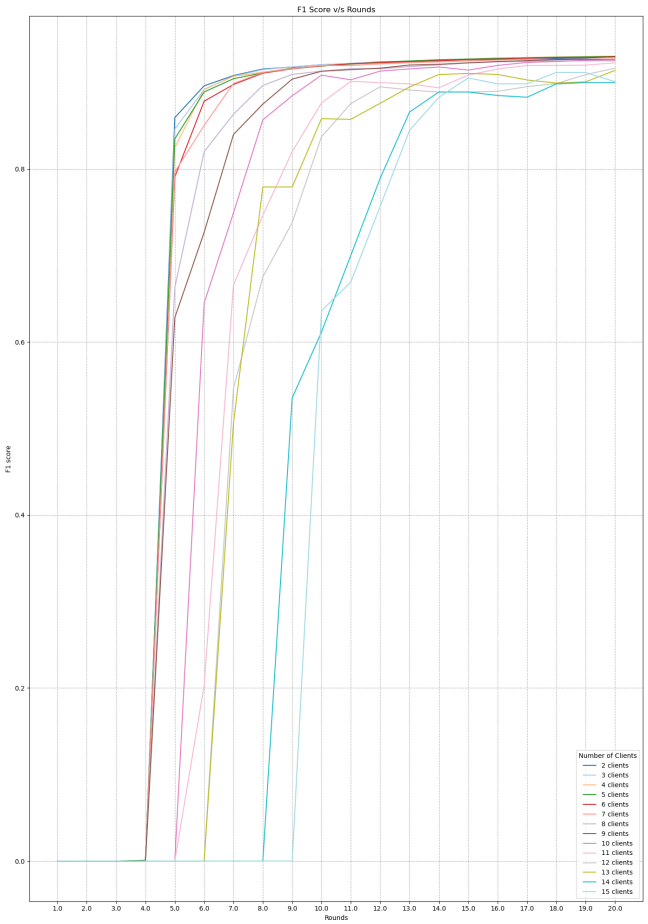


**FIGURE 11.** F1 Score v/s No. of rounds for different clients using Cyclic XGBoost Federated learning

As shown in Figure 11, the F1 Score improved as the number of rounds increases. This improvement was more rapid in the initial rounds (1-13) than in the later rounds (14-20). The F1 score varies significantly with the number of clients. For fewer clients (e.g., two or three clients), the F1 score starts to decrease, but improves more steadily over the rounds. As the number of clients increases, the initial F1 score starts to increase, and the improvement tends to be faster in the early rounds, but stabilizes sooner.

## XII. DETECTION OF BANK TRANSACTION ANOMALIES USING GRADIENT BOOSTED FEDERATED LEARNING WITH BOOTSTRAP AGGREGATION

Bootstrap Aggregation is also known as bagging in ensemble meta-learning algorithms in machine learning. It was developed primarily to improve the stability and accuracy of ML algorithms, particularly decision trees. The fundamental idea behind bagging is to reduce variance by averaging multiple models trained on different subsets of the training data. We performed bagging on the XGBoost algorithm to parallelize our process and streamline the algorithm for the detection of bank transaction anomalies. This can be achieved because each model was trained independently.

Specifically, each client acts as a bootstrap by performing random subsampling (i.e., data partitioning in FL). In every FL round, all clients boost a set of trees using their local bootstrap samples. The client trees are then combined on the server and appended to the global model from the previous round. This combined tree ensemble on the server was then considered the new global model as shown in Figure 12 A detailed algorithmic implementation is presented below.

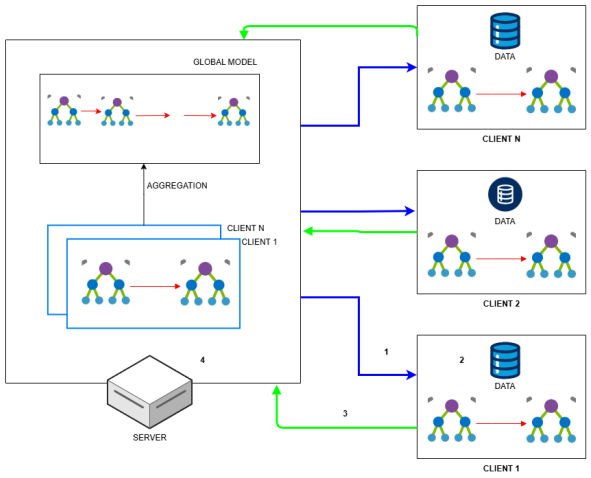


**FIGURE 12.** Federated Learning using XGBoost with Bagging aggregation

### A. DATA PARTITIONING

The dataset was divided among multiple clients, with each client having a distinct subset of data. This is crucial for simulating a real-world federated learning scenario, in which data are inherently distributed across different entities.

#### 1) Partitioning Strategy

The dataset was partitioned to ensure that each client had an approximately equal number of samples, while maintaining the original data distribution as much as possible. This partitioning was implemented in Python script, which includes functions for loading and partitioning the dataset. The data were loaded and split into multiple parts, each assigned to a different client.

#### 2) Local Data Storage

Each client stores its subset of data locally, ensuring that the raw data never leaves the client, thereby preserving privacy. Local storage setup is critical for maintaining data privacy and security. The data loading and partitioning functions handle the distribution of data to clients.

### B. CLIENT SETUP

Each client must be set up to perform local training on its data subset. The setup includes initializing the local environment and preparing the data for training.

#### 1) Environment Initialization

Each client set up its local environment, including installing the necessary libraries and configuring training parameters. The utility functions for these tasks were defined using Python scripts. These functions handle tasks, such as initializing the client, setting up communication protocols, and configuring the training parameters.

#### 2) Data Splitting for Validation

The local dataset was split into training and validation datasets. This ensured that each client could train and evaluate the model using separate data, thereby improving the generalizability of the model. The data preparation steps were defined in the `dataset.py` script, which ensured that each client had the necessary data for training and evaluation.

### C. FEDERATED LEARNING PARAMETERS INITIALIZATION

Before the federated learning process begins, several parameters must be initialized on both the server and client sides.

#### 1) Server-Side Initialization

- **Model Parameters:** An initial global model was created using the default XGBoost parameters. This model served as the starting point for federated training. The initial model parameters were defined in the server-setup script.
- **Server Configuration:** The server is configured to manage the federated learning process, including setting the number of rounds (epochs), aggregation strategy (e.g., weighted averaging), and communication protocols. The server setup and functions for managing clients and aggregating models are defined in `server.py` and `server_utils.py`. These scripts address the initialization of the server, configuration of communication protocols, and setup of aggregation strategies.

#### 2) Client-Side Initialization

- **Model Configuration:** Each client configures its local model using the initial parameters received from the server. The model includes hyperparameters specific to XGBoost, such as learning rate, maximum depth, and number of estimators. The model configuration was managed in a `client.py` script, where each client initialized its local model using the parameters received from the server.
- **Communication Setup:** Clients were configured to connect to the server using specified protocols. Each client prepares to send and receive the model parameters during the training process. The client setup and training functions are defined in `client.py`, which includes the configuration of the communication protocols and setup for parameter exchange with the server.

### D. ALGORITHM

The federated learning process involves multiple rounds of communication between the server and the clients. Each round consisted of the following steps.

#### 1) Server sends model to Clients

At the beginning of each round, the server sends the current global model parameters to all clients. This is orchestrated using server functions that handle client communication. The server uses a communication protocol to send current global model parameters to each client. This ensured synchronization across all clients at the start of each training round. The Python script server includes functions for broadcasting the model parameters to all clients.

#### 2) Clients Perform Local Training

Upon receiving the global model, each client performs the following tasks:

- **Data Preparation:** Clients loaded their local training and validation data, as managed by the Python script dataset.
- **Model Training:** Clients train their local XGBoost model using training data. The training process involved multiple iterations, in which the model was updated using gradient boosting. Each client used its data to compute the gradients and updated the model parameters accordingly.
- **Gradient Boosting:** For iterations, the model is updated by adding new trees that predict the residuals (errors) of the current tree ensemble. The objective function, including both the loss function and the regularization term, was optimized using the gradient descent method. The client Python script handles the training process, in which each client updates its local model using its data.
- **Model Evaluation:** After training, clients evaluate the updated model using their local validation data. Metrics, such as accuracy, precision, recall, and F1-score, were computed to assess the model's performance. The evaluation process is part of the client-side implementation in the Python script client.py, where each client evaluates its model and prepares evaluation metrics for the server.

#### 3) Client sends model updates to Server

After completing the local training and evaluation, each client sent the updated model parameters and evaluation metrics back to the server. This is facilitated through client-server communication functions defined in the client and server utility Python scripts. Each client uses a communication protocol to send the updated parameters and performance metrics to the server. This step is critical for ensuring that the server receives accurate updates from all clients. The client Python script includes functions for sending the updated parameters and metrics to the server.

#### 4) Server Aggregates Model Updates

The server aggregates the model updates received from all the clients. The aggregation process involves the following steps.

- **Weighted Averaging:** The server computes the weighted average of the model parameters from all

clients. The weights were proportional to the number of samples used for training for each client. Mathematically, if $\theta_i$ represents the parameters from client $i$ and $n_i$ the number of samples, and the global parameter $\theta$ is computed as

$$\theta = \frac{\sum_{i=1}^{N} n_i \theta_i}{\sum_{i=1}^{N} n_i} \tag{20}$$

- **Update the Global Model:** The Aggregated parameters are used to update the global model. This updated model is sent to clients in the next round. The `server.py` script handles the aggregation process in which the server computes the weighted average of the model parameters and updates the global model.

#### 5) Round Iteration

This process is then iterated based on the hardware capability of the user as well as the desired performance metrics.

### E. RESULTS

The XGBoost Federated Learning with Bootstrap Aggregation model was run with different numbers of clients for up to 20 rounds, and the performance metrics were captured at the end of each round. The metrics obtained for this model are as follows:

#### 1) Time taken to run model

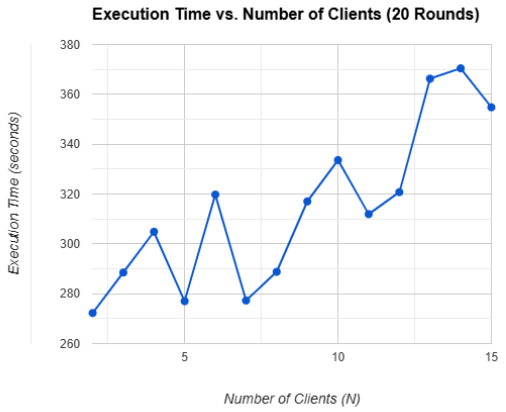


**FIGURE 13.** Time taken to run the Bagging Model v/s Number of clients

Graph 13 indicates a general upward trend in the time taken to execution of twenty rounds as the number of clients increases. The execution time started at approximately 270 s for two clients and increased to approximately 350-370 seconds for 15 clients. The execution time did not increase linearly, but fluctuated at various points.

These fluctuations suggest that there may be factors other than the number of clients affecting execution time, such as communication overhead, client performance variability, or network latency. The increase in execution time with the number of clients indicates that the federated learning process becomes more resource intensive as more clients participate.

#### 2) Confusion Matrix

| No. of Clients | True Positives | False Positives | True Negatives | False Negatives |
|---|---|---|---|---|
| 2 | 956460 | 4881 | 7233 | 88463 |
| 3 | 956144 | 5197 | 6806 | 88890 |
| 4 | 955960 | 5381 | 6425 | 89271 |
| 5 | 955688 | 5653 | 6352 | 89344 |
| 6 | 955374 | 5967 | 6381 | 89315 |
| 7 | 955420 | 5921 | 6375 | 89321 |
| 8 | 955242 | 6099 | 6473 | 89223 |
| 9 | 955409 | 5932 | 6692 | 89004 |
| 10 | 955323 | 6018 | 6470 | 89226 |
| 11 | 955037 | 6304 | 6349 | 89347 |
| 12 | 955584 | 5757 | 6574 | 89122 |
| 13 | 955310 | 6031 | 6434 | 89262 |
| 14 | 955765 | 5576 | 6915 | 88781 |
| 15 | 955530 | 5811 | 6724 | 88972 |

**TABLE 4.** Confusion Matrix Values for Different Number of Clients in Bagging Configuration

Table 4 presents the confusion matrices for different numbers of clients obtained after 20 rounds of training using XGBoost Federated Learning with Bagging.

#### 3) Accuracy

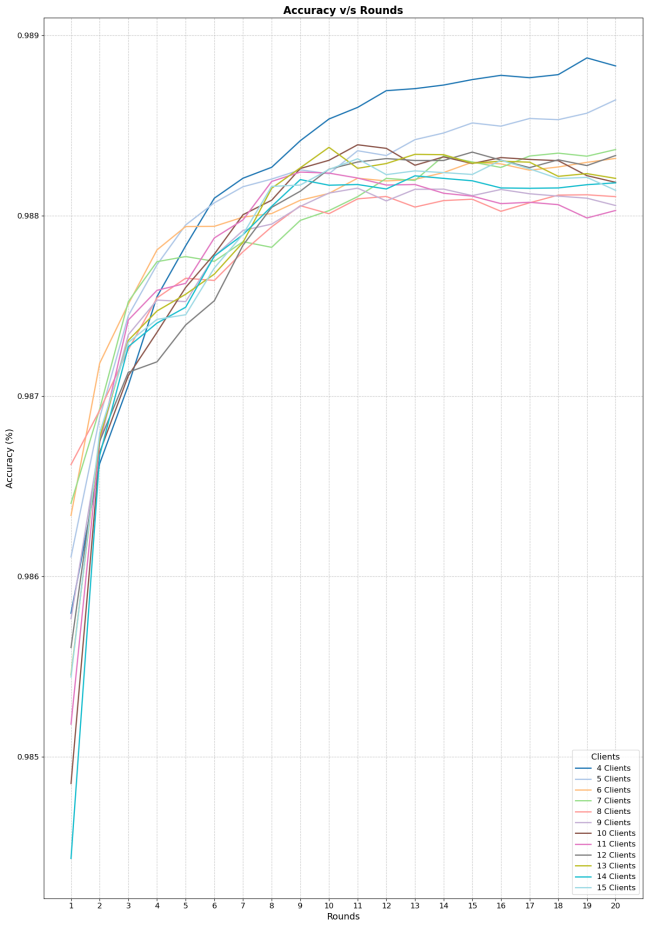


**FIGURE 14.** Accuracy v/s No. of rounds for different clients using XGBoost Federated Learning with Bagging

Fig. 14 illustrates the accuracy of the model over different rounds. Most client configurations showed minor variations, with the accuracy remaining relatively stable after the initial increase.

The model accuracy improved significantly in the early rounds (1-4), indicating effective initial learning. The rapid increase in accuracy suggests that the federated learning process quickly enhanced the model's performance. After the initial rounds, the accuracy stabilized and remained high, with minor fluctuations, demonstrating consistent performance across rounds (5-20).

The number of clients did not significantly affect the final accuracy, as all configurations achieved high accuracy by round four and maintained it throughout the subsequent rounds.

### 4) Precision

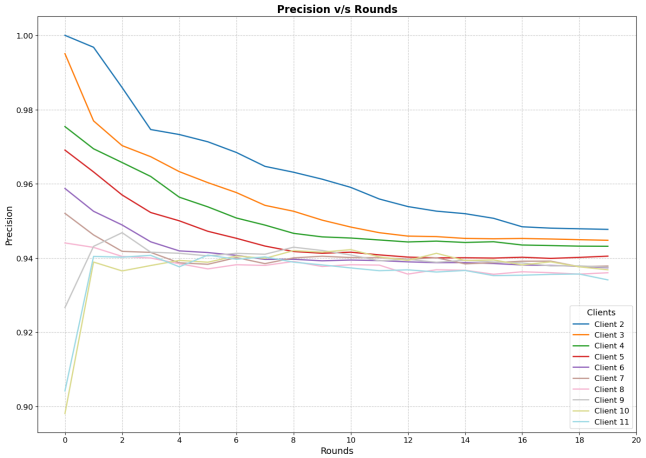


**FIGURE 15. Precision v/s No. of rounds for different clients using XGBoost Federated Learning with Bagging**

Figure 15 illustrates the precision of the model over different rounds. There was a general decline in precision as the number of rounds increased. The decline was more pronounced in the initial rounds (1-4) and continued at a slower pace in later rounds (5-20).

For configurations with fewer clients (e.g., Clients 2 and 3), the precision starts to be very high (close to 1.0) and declines steadily. Configurations with more clients (e.g., Clients 14 and 15) started with lower precision and showed a gradual decline.

The initial rounds (1-4) were critical and exhibited the most significant decline in precision. After these rounds, the decline slowed, indicating that the performance of the model stabilized.

### 5) Recall

Figure 16 illustrates the recall of the model over different rounds. The recall generally improved as the number of rounds increased. The improvement is more rapid in the initial rounds (1-4) and continues at a slower pace, but more steady in the later rounds (5-20).

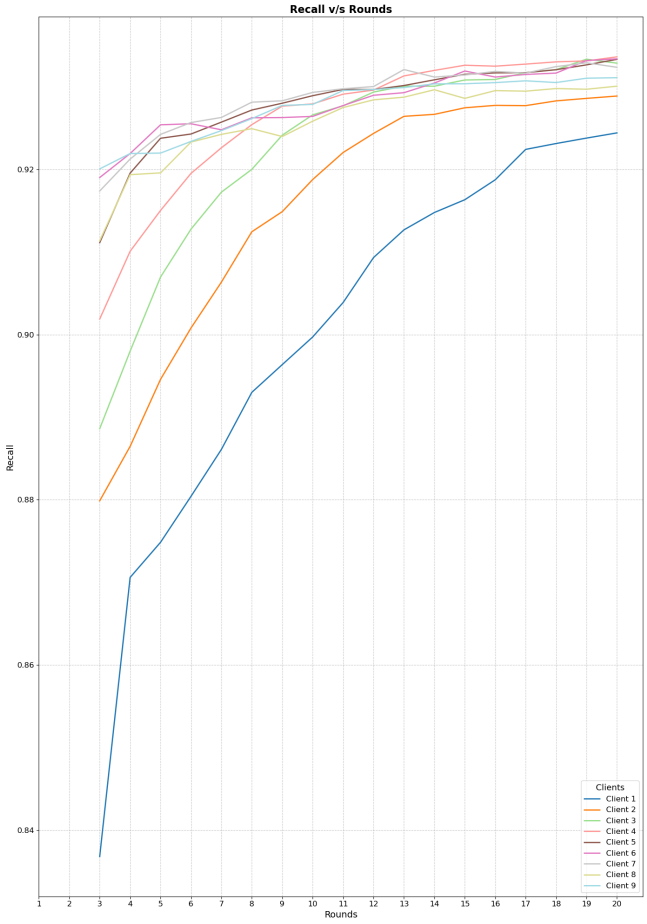


**FIGURE 16. Recall v/s No. of rounds for different clients using XGBoost Federated Learning with Bagging**

For configurations with fewer clients (e.g., Client 1), recall starts lower, but improves rapidly over the rounds. Configurations with more clients (e.g., Clients 2 and 3) start with higher recall and show a steadier improvement.

This suggests that the federated learning process effectively increased the model's ability to correctly identify positive instances.

### 6) F1 Score

Graph 17 illustrates the F1 Score of the model over different rounds segmented into two ranges: (1-4) and (5-20). In the graph for rounds 1-13, there was a noticeable improvement in the F1 scores as the number of rounds increased. The F1 score starts from lower values and generally increases; the values tend to converge and remain relatively high, indicating consistent performance and showing that the model's balance between precision and recall improved over time.

## XIII. ANALYSIS

This section provides a detailed overview of the analysis and inferences that we draw upon while comparing our federated learning setup with current existing conventional machine learning techniques.

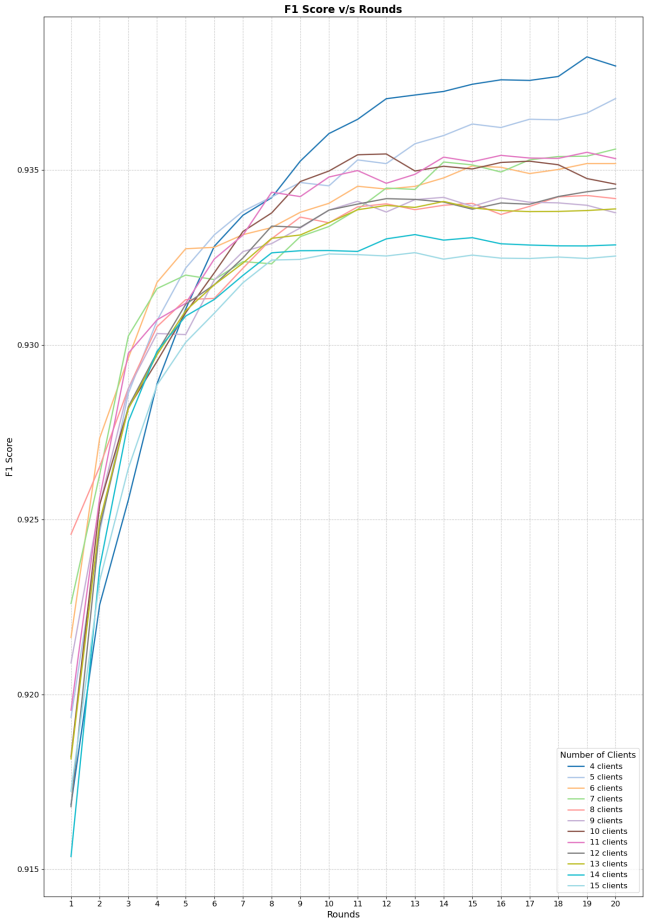


**FIGURE 17.** F1 Score v/s No. of rounds for different clients using XGBoost Federated Learning with Bagging

### A. COMPARATIVE PERFORMANCE METRICS ACROSS MODELS

To evaluate the efficacy of our proposed Cyclic and Bagging Federated Learning (FL) models, we compared their performance against traditional centralized XGBoost, Random Forests, and Deep Learning-based methods. The comparison spans key evaluation metrics: Accuracy, Precision, Recall, and F1 Score are shown in Graph 18.

The centralized XGBoost model demonstrates strong performance across metrics; however, its reliance on centralized data aggregation raises privacy and scalability concerns. Random Forests exhibit competitive results, but they struggle with high-dimensional data and class imbalance, particularly evident in lower Recall and F1 scores. Deep Learning models, while robust in complex pattern recognition, face challenges such as overfitting and high computational demands, especially in imbalanced datasets like ours.

In contrast, the Cyclic FL XGBoost model significantly outperforms traditional methods across all metrics. It achieves an impressive accuracy of 98% and recall of 97%, reflecting its ability to generalize across distributed datasets while preserving data privacy. Similarly, the Bagging FL XGBoost model, with an accuracy of 97% and recall of 96%, balances

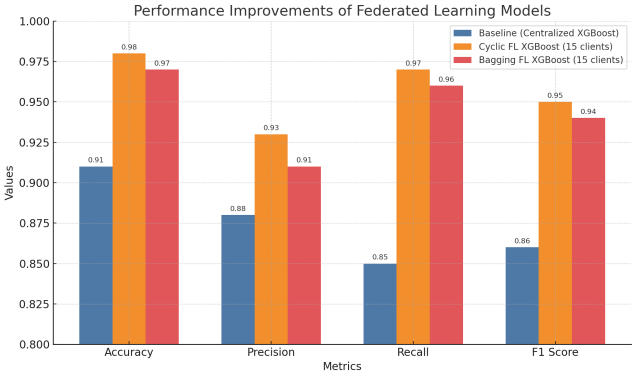


**FIGURE 18.** Comparative Performance with Baseline Centralized XGBoost

computational efficiency and performance. These results underscore the strength of federated learning techniques in handling imbalanced datasets and privacy-preserving environments as shown in Graph 19.


**FIGURE 19.** Comparative Performance Metrics Across Different Models

### B. ERROR ANALYSIS ACROSS MODELS

To further assess model performance, we analyzed False Positives (FP) and False Negatives (FN). False Positives represent legitimate transactions flagged as fraudulent, while False Negatives indicate fraudulent transactions missed by the model.

The centralized XGBoost and Random Forest models exhibited higher FP and FN values, with the latter being especially detrimental in financial fraud detection scenarios Table 5. For instance, centralized XGBoost reported 4,881 FP and 88,463 FN, while Random Forests showed even higher FN values due to their limited handling of minority-class instances. Deep Learning models, although slightly better in FP reduction, still struggled with FN reduction due to imbalanced dataset challenges.

The Cyclic FL XGBoost model showed remarkable improvement, with FN reduced to 80,486 and FP to 2,466. Similarly, Bagging FL XGBoost achieved FN of 88,972 and FP of 5,811. These reductions in FN highlight the ability of

| Models | False Positives | False Negatives |
|---|---|---|
| Centralized XGBoost | 4881 | 88463 |
| Random Forests | 5300 | 89000 |
| Deep Learning | 4700 | 87000 |
| Cyclic FL | 2466 | 80486 |
| Bagging FL | 5811 | 88972 |

**TABLE 5.** Error Analysis Across Models

federated learning approaches to detect anomalies effectively while minimizing misclassifications, especially crucial for preserving customer trust and financial security.

## XIV. CONCLUSION OF THE STUDY

In this penultimate section, we present a collated conclusion of our unique approach and solution to the current privacy and security challenges faced by anomaly-detection algorithms used in financial institutions.

The implementation of our Cyclic and Bagging Federated Learning (FL) XGBoost models demonstrates strong scalability and communication efficiency, making them practical for real-world distributed environments like financial institutions and banking networks. These characteristics are particularly relevant given the decentralized nature of banking operations, where data resides across multiple branches or entities.

### A. SCALABILITY AND ADAPTABILITY

Scalability is a critical factor in distributed systems, particularly in environments where the volume of transactions and data grows continuously. Our models were tested with varying numbers of participating clients (from 2 to 15) across multiple rounds to evaluate scalability.

#### 1) Time Efficiency

The execution time for both Cyclic and Bagging FL models increased as the number of clients grew but did so in a manageable manner. For example, the Cyclic FL model scaled from 240 seconds with 2 clients to approximately 370 seconds with 15 clients. The Bagging FL model followed a similar trend but required slightly more time due to the inherent computational overhead of aggregating multiple trees in each round.

#### 2) Performance Trade-Offs

While accuracy and recall remained consistent as the number of clients increased, precision showed minor fluctuations due to the limited size of the dataset distributed among clients. These results suggest that the models are well-suited for larger networks but would benefit from adequate data distribution to maintain performance.

#### 3) Client Contributions

Both Cyclic and Bagging FL approaches ensure that each client contributes uniquely to the global model, effectively integrating diverse data distributions from across the network. This adaptability enhances the model's robustness, particularly in heterogeneous environments.

### B. COMMUNICATION EFFICIENCY

Efficient communication between clients and the central server is of utmost importance in federated learning systems to minimize latency and bandwidth usage.

#### 1) Reduced Data Transmission

Our models transmit only model parameters (e.g., gradients and tree updates) rather than raw data. This approach drastically reduces the amount of data transferred, ensuring compliance with privacy regulations while maintaining communication efficiency.

#### 2) Incremental Updates

The Cyclic FL model employs a sequential training approach where each client incrementally updates the model. This minimizes the computational burden on the server during each round, making it suitable for low-bandwidth environments.

#### 3) Parallelization in Bagging

The Bagging FL model enables parallel training across clients, aggregating updates at the server. Although this increases communication overhead compared to Cyclic FL, the parallelization significantly speeds up the training process, making it ideal for environments with robust network infrastructure.

### C. IMPLICATIONS FOR FINANCIAL INSTITUTIONS

Our study provides a comprehensive framework for financial institutions to implement decentralized privacy-preserving anomaly detection systems by demonstrating that Cyclic and Bagging FL XGBoost models are not only accurate and robust but also scalable and efficient for real-world distributed environments. These attributes make them a valuable tool for financial institutions aiming to enhance fraud detection while ensuring data privacy and compliance with regulatory standards. Our findings also emphasize the importance of choosing an appropriate federated learning approach based on the specific needs and resource availability of the institution.

## XV. FUTURE SCOPE

The study of gradient-boosted federated learning for detecting bank transaction anomalies provides a strong foundation for future research and development. This section outlines the scope and potential of this research, emphasizing advanced techniques and expanding the applicability of federated learning in the financial and other data domains.

### A. VERTICAL FEDERATED LEARNING

Vertical federated learning involves institutions with different feature spaces but overlapping samples collaborate to train a model. Vertical federated learning enables the integration

of diverse feature sets, enriching the model with more comprehensive insights into customer behaviour [39]. This collaborative approach can lead to more accurate and holistic anomaly detection models by utilizing the synergy between the different feature spaces.

### B. MULTIMODAL FEDERATED LEARNING

Different financial institutions may use different types of machine learning models tailored to their specific data and requirements. Federated learning frameworks can be designed to aggregate these heterogeneous models to create a more comprehensive and accurate global model [40].

The federated learning framework can be extended to support aggregation of various model architectures. This heterogeneity enhanced the robustness and generalizability of the global model.

## ACKNOWLEDGMENT

We extend our heartfelt gratitude to PES University for their unwavering support and encouragement throughout our research. Their generosity in providing us with the latest computational resources has been invaluable in facilitating the execution of our machine learning models. Access to these resources contributed significantly to the successful completion of this study. We appreciate the university's commitment to fostering an environment that promotes academic excellence and innovation.

## REFERENCES

[1] R. T. Stewart, "Bank fraud and the macroeconomy," *Journal of Operational Risk*, vol. 11, no. 1, 2016.

[2] W. H. Muller, C. H. Kalin, and J. G. Goldsworth, *Anti-money laundering: international law and practice*. John Wiley & Sons, 2007.

[3] A. A. S. Alsuwailem and A. K. J. Saudagar, "Anti-money laundering systems: a systematic literature review," *Journal of Money Laundering Control*, vol. 23, no. 4, pp. 833–848, 2020.

[4] G. Xu, H. Li, H. Ren, K. Yang, and R. H. Deng, "Data security issues in deep learning: Attacks, countermeasures, and opportunities," *IEEE Communications Magazine*, vol. 57, no. 11, pp. 116–122, 2019.

[5] S. K. Lo, Q. Lu, H.-Y. Paik, and L. Zhu, "Flra: A reference architecture for federated learning systems," in *European Conference on Software Architecture*. Springer, 2021, pp. 83–98.

[6] Y. Zhang, J. Tong, Z. Wang, and F. Gao, "Customer transaction fraud detection using xgboost model," in *2020 International Conference on Computer Engineering and Application (ICCEA)*. IEEE, 2020, pp. 554–558.

[7] R. J. Bolton, D. J. Hand *et al.*, "Unsupervised profiling methods for fraud detection," *Credit scoring and credit control VII*, pp. 235–255, 2001.

[8] R. K. Gopal and S. K. Meher, "A rule-based approach for anomaly detection in subscriber usage pattern," in *Proceedings of World Academy of Science, Engineering and Technology*. Citeseer, 2007, pp. 396–399.

[9] Q. Zhang, "Financial data anomaly detection method based on decision tree and random forest algorithm," *Journal of Mathematics*, vol. 2022, no. 1, p. 9135117, 2022.

[10] M. Hejazi and Y. P. Singh, "One-class support vector machines approach to anomaly detection," *Applied Artificial Intelligence*, vol. 27, no. 5, pp. 351–366, 2013.

[11] D. Huang, D. Mu, L. Yang, and X. Cai, "Codetect: Financial fraud detection with anomaly feature detection," *IEEE Access*, vol. 6, pp. 19 161–19 174, 2018.

[12] M. Zamini and G. Montazer, "Credit card fraud detection using autoencoder based clustering," in *2018 9th International Symposium on Telecommunications (IST)*. IEEE, 2018, pp. 486–491.

[13] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[14] V. Mothukuri, P. Khare, R. M. Parizi, S. Pouriyeh, A. Dehghantanha, and G. Srivastava, "Federated-learning-based anomaly detection for iot security attacks," *IEEE Internet of Things Journal*, vol. 9, no. 4, pp. 2545–2554, 2021.

[15] W. Yang, Y. Zhang, K. Ye, L. Li, and C.-Z. Xu, "Ffd: A federated learning based method for credit card fraud detection," in *Big Data–BigData 2019: 8th International Congress, Held as Part of the Services Conference Federation, SCF 2019, San Diego, CA, USA, June 25–30, 2019, Proceedings 8*. Springer, 2019, pp. 18–32.

[16] A. O. Hoffmann and C. Birnbrich, "The impact of fraud prevention on bank-customer relationships: An empirical investigation in retail banking," *International journal of bank marketing*, vol. 30, no. 5, pp. 390–407, 2012.

[17] Y. Kou, C.-T. Lu, S. Sirwongwattana, and Y.-P. Huang, "Survey of fraud detection techniques," in *IEEE international conference on networking, sensing and control, 2004*, vol. 2. IEEE, 2004, pp. 749–754.

[18] J. Serrado, R. F. Pereira, M. Mira da Silva, and I. Scalabrin Bianchi, "Information security frameworks for assisting gdpr compliance in banking industry," *Digital Policy, Regulation and Governance*, vol. 22, no. 3, pp. 227–244, 2020.

[19] K. Patel, "Credit card analytics: a review of fraud detection and risk assessment techniques," *International Journal of Computer Trends and Technology*, vol. 71, no. 10, pp. 69–79, 2023.

[20] E. Lopez-Rojas, A. Elmir, and S. Axelsson, "Paysim: A financial mobile money simulator for fraud detection," in *28th European Modeling and Simulation Symposium, EMSS, Larnaca*. Dime University of Genoa, 2016, pp. 249–255.

[21] E. A. Lopez-Rojas and C. Barneaud, "Advantages of the paysim simulator for improving financial fraud controls," in *Intelligent Computing: Proceedings of the 2019 Computing Conference, Volume 2*. Springer, 2019, pp. 727–736.

[22] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.

[23] S. Banabilah, M. Aloqaily, E. Alsayed, N. Malik, and Y. Jararweh, "Federated learning review: Fundamentals, enabling technologies, and future applications," *Information processing & management*, vol. 59, no. 6, p. 103061, 2022.

[24] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[25] T. Sun, D. Li, and B. Wang, "Decentralized federated averaging," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 4, pp. 4289–4301, 2022.

[26] X. Liu, L. Xie, Y. Wang, J. Zou, J. Xiong, Z. Ying, and A. V. Vasilakos, "Privacy and security issues in deep learning: A survey," *IEEE Access*, vol. 9, pp. 4566–4593, 2020.

[27] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," *arXiv preprint arXiv:2007.05558*, vol. 10, 2020.

[28] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019.

[29] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM computing surveys (CSUR)*, vol. 54, no. 2, pp. 1–38, 2021.

[30] F. Giannakas, C. Troussas, A. Krouska, C. Sgouropoulou, and I. Voyiatzis, "Xgboost and deep neural network comparison: The case of teams' performance," in *Intelligent Tutoring Systems: 17th International Conference, ITS 2021, Virtual Event, June 7–11, 2021, Proceedings 17*. Springer, 2021, pp. 343–349.

[31] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[32] D. Nielsen, "Tree boosting with xgboost-why does xgboost win" every" machine learning competition?" Master's thesis, NTNU, 2016.

[33] Y.-Y. Song and L. Ying, "Decision tree methods: applications for classification and prediction," *Shanghai archives of psychiatry*, vol. 27, no. 2, p. 130, 2015.

[34] G. Velarde, A. Sudhir, S. Deshmane, A. Deshmunkh, K. Sharma, and V. Joshi, "Evaluating xgboost for balanced and imbalanced data: application to fraud detection," *arXiv preprint arXiv:2303.15218*, 2023.

[35] S. B. S. Lai, N. Shahri, M. B. Mohamad, H. Rahman, and A. B. Rambli, "Comparing the performance of adaboost, xgboost, and logistic regression for imbalanced data," *Mathematics and Statistics*, vol. 9, no. 3, pp. 379–385, 2021.
[36] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão *et al.*, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.
[37] ——, "Flower: A friendly federated learning framework," 2022.
[38] X. Zhao, X. Li, S. Sun, and X. Jia, "Secure and efficient federated gradient boosting decision trees," *Applied Sciences*, vol. 13, no. 7, p. 4283, 2023.
[39] T. Chen, X. Jin, Y. Sun, and W. Yin, "Vafl: a method of vertical asynchronous federated learning," *arXiv preprint arXiv:2007.06081*, 2020.
[40] L. Che, J. Wang, Y. Zhou, and F. Ma, "Multimodal federated learning: A survey," *Sensors*, vol. 23, no. 15, p. 6986, 2023.

**ROHAN CHANDRASHEKAR** is currently pursuing Bachelor of Technology degree (B.Tech) in Computer Science and Engineering (CSE) from PES University, Bengaluru, India.

He is an intern at Hewlett Packard Enterprise (HPE), working as part of the cybersecurity team as a Technology Consultant.

He is currently working on HPE's customer self service portal SQuest, on top of Ansible automation platform that offers Everything as a Service (EaaS).

He has conducted lightning talks and published multiple research papers in leading conferences, one of which has won Best Paper Award at the IEEE International MRTM Conference 2023. His research interests include Machine Learning (ML), Artificial Intelligence (AI), Quantum Computing, Natural Language Processing (NLP) and Cybersecurity.

**RITHVIK GRANDHI** is currently pursuing Bachelor of Technology degree (B.Tech) in Computer Science and Engineering (CSE) from PES University, Bengaluru, India.

He is an intern at the 5G Testbed, IIT Madras, and works as part of the software team.

His research interests include Machine Learning (ML), Artificial Intelligence (AI) , computer vision and cloud computing.

**RAHUL ROSHAN G** is currently pursuing Bachelor of Technology degree (B.Tech) in Computer Science and Engineering (CSE) from PES University, Bengaluru, India.

He is an intern at the Information Security, Forensics and Cyber Resilience (PESU-ISFCR), PES University. He worked as a research intern on the cloud security team.

He has published multiple research papers in leading conferences including International Conference for Emerging Technology 2024 and won Best Paper Award at the IEEE International MRTM Conference 2023. His research interests include Machine Learning (ML), Artificial Intelligence (AI) and Cloud security.

**DR. SHYLAJA S SHARATH** is the Director, Centre for Cloud Computing and Big Data and Centre for Data Science and Applied Machine Learning, Former Chairperson, Department of Computer Science and Engineering, PESU,& Professor & Head of CSE & ISE, PESIT

She completed her Bachelor's degree from UVCE, Bangalore University in 1989, Master's degree from Sri Jayachamarajendra College of Engineering, Mysore University in 1993. She secured the First rank in the university in her M. Tech course. She completed the "C" level course from DOEACC and holds a Ph.D in the domain of Face Recognition from Visvesvaraya Technological University (VTU). She has 28 years of teaching experience and 15 years of research experience.

She has several journal articles and national and international conference publications to her credit. She has taught a plethora of courses and guided several projects and research activities at the undergraduate and postgraduate levels. She is a member of the BOS of several organizations, conferences, and members of many technical committees. Dr. Shylaja is heading two centers at P E S University, the Center for Data Sciences and Applied Machine Learning (CDSAML) and the Huawei Innovation Lab, facilitating research and internship opportunities for students and faculty members. Her research areas include image processing, computer vision, natural language processing, and machine learning.