

# A Federated Learning Framework For Disease Prognosis Using Tree Ensembles.

Abdul Rehman Mohsin  
Saad Rasheed

January 20, 2025

## Abstract

Federated learning is an emerging paradigm in machine learning that enables the training of models across multiple decentralized devices or servers holding local data samples, without sharing their data. This approach ensures data remains on local devices, addressing privacy concerns for sensitive information like healthcare data while avoiding the cost and storage inefficiencies of centralizing decentralized data. Further highlighting the importance of this approach. Federated learning is categorized into two types: horizontal federated learning, where datasets between participants share the same feature space but differ in samples, and vertical federated learning, where participants have different feature spaces with overlapping samples. This work focuses on horizontal federated learning. A primary challenge in federated learning is handling non-independent and identically distributed (Non-IID) data. Since data across different nodes often reflects varying distributions, this non-uniformity can adversely affect model convergence and overall performance. This work develops a federated learning framework addressing the outlined challenges. A federated learning algorithm is proposed, with its results evaluated and compared to centralized training and baseline federated approaches. Additionally, a secure communication protocol using SSL/TLS ensures the integrity and confidentiality of communications between nodes. To preserve data anonymity, a method for aggregating updates that masks the client updates from a potential malicious server is also introduced.

## 1. Introduction

Federated learning is an emerging approach in machine learning that enables collaborative model training across multiple nodes while keeping data localized. This approach addresses privacy concerns by ensuring that data remains on the local device, making it particularly suitable for scenarios involving highly sensitive information such as healthcare and medical data. Additionally, gathering decentralized data into a central repository can be cost-inefficient and pose significant storage challenges. Federated learning can be broadly categorized into two types: horizontal federated learning, where datasets share the same feature space but differ in samples, and vertical federated learning, where datasets have overlapping samples but distinct feature spaces.

This work focuses on horizontal federated learning, which is particularly applicable when participants possess datasets with similar structures but different entries.

A key assumption in traditional machine learning is that data is independently and identically distributed (IID). The machine learning objective function can be defined as follows:

$$\min_{w \in R^d} f(w), \quad \text{where } f(w) = \frac{1}{n} \sum_{i=1}^n f_i(w)$$

In federated learning, assume there are  $K$  nodes over which the data is partitioned. Let  $P_k$  be the set of indices of data points on node  $k$ , and  $n_k = |P_k|$ . The global objective can then be rewritten as:

$$f(w) = \sum_{k=1}^K \frac{n_k}{n} F_k(w), \quad \text{where } F_k(w) = \frac{1}{n_k} \sum_{i \in P_k} f_i(w)$$

The IID assumption implies:

$$E_{P_k}[F_k(w)] = f(w)$$

Under the Non-IID assumption,  $F_k$  could be an arbitrarily poor approximation of  $f$ , posing significant challenges for federated learning. Addressing the impact of Non-IID data is a primary focus of this work.

## 2. Literature Review

Federated learning has gained significant attention in recent years due to its potential for privacy-preserving collaborative learning. A foundational algorithm in this domain is FedAvg, introduced by McMahan et al. [1]. FedAvg combines local updates from participating nodes to create a global model, making it an effective and communication-efficient approach for deep learning in federated settings.

Despite its success, practical work on horizontal federated learning for tree-based models remains scarce. The main obstacle in horizontal federated learning is the Non-IID nature of data, which hinders model convergence and performance. Several studies have addressed this challenge. For instance, McMahan et al. [1] provide a comprehensive definition of the Non-IID assumption and propose FedAvg as a solution.

Chen and Chao [2] introduce FEDBE, a Bayesian model ensemble approach that leverages the Monte Carlo method to approximate ensemble models in federated learning. This method enhances the robustness of federated learning under Non-IID conditions.

In the context of tree-based methods, Ong et al. [3] propose an adaptive histogram-based gradient-boosted trees approach for federated learning. Their method effectively addresses privacy concerns and the Non-IID challenge. Building on this, Lindskog et al. [4] improve the histogram-based approach by introducing minimal variance sampling, further enhancing the performance of federated learning on tabular data.

Security is another critical aspect of federated learning. Updates must be secure to prevent leakage of sensitive information. Zhu et al. [5] demonstrate that gradients can be exploited to reconstruct training data in reconstruction attacks. Hitaj et al. [6] show that Generative Adversarial Networks (GANs) can replicate data with similarities to training samples in collaborative setups. Carlini et al. [7] highlight how language models can memorize and leak confidential training data, emphasizing the need for robust security measures in federated learning.

### 3. Methodology

The starting point was a modeling pipeline utilizing a feature-engineered dataset. The disease prognosis data was divided by state and distributed across different local servers. A test set comprising 10% of the dataset was separated, ensuring state-wise stratification to maintain proportional representation.

#### 3.1. Phase 1: A Federated Machine Learning Algorithm

In this phase, several baseline approaches to implementing federated learning with tree-based models were explored.

##### 3.1.1 Baseline Approaches

**1. Weighted Ensemble of Tree-based Models:** This approach involves combining multiple decision tree models using an ensemble method, where each model is weighted according to certain criteria, such as the performance or contribution of each individual model. The idea is to leverage the diversity of different models to improve overall performance.

**2. Cyclic XGBoost:** This technique involves sequentially training boosting trees on each client. In cyclic XGBoost, the model goes through multiple iterations of training on different clients, effectively allowing each node to refine its contribution to the global model over time. The sequential approach helps in optimizing the overall performance by making use of a cyclic updating mechanism.

##### Weighted vs. Unweighted Cyclic XGBoost:

- **Weighted Approach:** The number of trees assigned to each client is proportional to the number of samples present at that node. Clients with more data will have a larger number of trees, thus contributing more to the global model. This allows the model to handle varying data distributions across nodes.
- **Unweighted Approach:** All clients train the same number of trees, regardless of the amount of data at each node. This approach assumes that each client's data is of equal importance, regardless of the size.

##### 3.1.2 Problems Identified

Despite efforts to optimize these initial approaches, several challenges arose:

- **Weighted Ensembling and Voted Aggregation:** Both the weighted ensemble and majority vote aggregation techniques exhibited poor performance. The challenges here stemmed from the imbalance in contributions from different models, and the aggregation methods failed to capture the underlying patterns effectively across nodes.
- **Cyclic XGBoost Overfitting:** The cyclic XGBoost model tended to overfit when it reached the last node in the sequence. This overfitting occurred because the model placed too much emphasis on the last iteration, resulting in poor generalization

to unseen data. Overfitting on sequential nodes can lead to suboptimal model performance when the global model is aggregated.

### 3.1.3 Solutions

To address these issues, several solutions were explored:

- **Tune Learning Rate and Number of Trees per Node:** One way to address these issues is to tune the hyperparameters, specifically the learning rate and the number of trees each client uses. Keeping low the number of trees or the learning rate ensures that the model gradually learns the patterns over the dataset instead of fitting on a single node.
- **Weighted Approach for Tree Allocation:** A more refined approach for distributing the number of trees across clients is to allocate them proportionally to the number of samples at each client. This ensures that nodes with more data contribute more significantly to the global model. This technique aims to improve the accuracy of the model by taking into account the data distribution across clients.

### 3.1.4 Main Challenges Still Persist

Despite these solutions, the following challenges continued to persist:

- **Scalability to a Large Number of Nodes:** A major concern with the current approach is that the number of estimators is proportional to the number of nodes in the system. In scenarios with hundreds or thousands of nodes (e.g., a federated learning network with 1,000 clients), this approach would require an impractically large number of trees, leading to significant computational overhead.
- **Non-IID Data:** Another key issue is the presence of non-independent and identically distributed (non-IID) data across clients. Since data on each client may be skewed or inconsistent with the data from other clients, this introduces challenges in ensuring that the global model generalizes well across all nodes. Federated learning systems often struggle with the heterogeneous nature of data, which can affect the accuracy and fairness of the model.

### 3.1.5 From XGBoost to FedXGBoost

In XGBoost, the prediction  $\hat{y}_i^{(t)}$  at boosting round  $t$  is given by:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

where  $f_t$  is the  $t$ -th boosting tree function, and  $x_i$  is the feature vector for the  $i$ -th sample. At each step, a tree is built to optimize the objective function:

$$\text{obj}^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^t w(f_i)$$

where  $l(y_i, \hat{y}_i^{(t)})$  is the loss function, and  $w(f_i)$  is the complexity of the  $i$ -th tree.

We can rewrite the objective at boosting round  $t$  as:

$$\text{obj}^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + w(f_t) + \text{constant}$$

Next, the loss function is approximated using a second-order Taylor expansion:

$$\text{obj}^{(t)} \approx \sum_{i=1}^n \left[ l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + w(f_t) + \text{constant}$$

where  $g_i$  and  $h_i$  are the first and second-order derivatives of the loss function:

$$g_i = \frac{\partial}{\partial \hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

$$h_i = \frac{\partial^2}{\partial (\hat{y}_i^{(t-1)})^2} l(y_i, \hat{y}_i^{(t-1)})$$

Thus, the objective function becomes:

$$\text{obj}^{(t)} = \sum_{i=1}^n \left[ g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + w(f_t)$$

The tree function  $f_t(x)$  is defined as:

$$f_t(x) = w_{q(x)}, \quad w \in R^T, \quad q : R^d \rightarrow \{1, 2, \dots, T\}$$

where  $T$  is the number of leaves in the tree, and  $q$  is a function that assigns each data point to its corresponding leaf.  $w$  is the vector of scores on the leaves. The regularization term  $w(f)$ , which represents the complexity of the tree, is defined as:

$$w(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

Thus, the objective becomes:

$$\text{obj}^{(t)} \approx \sum_{i=1}^n \left[ g_i w_q(x_i) + \frac{1}{2} h_i w_q(x_i)^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

Summing over the tree leaves instead of the samples, the objective may be written as:

$$\text{obj}^{(t)} = \sum_{j=1}^T \left[ G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T$$

where  $G_j$  and  $H_j$  are gradients and Hessians aggregated over all the samples in the  $j_{th}$  leaf. Further, the score  $w_j$  is defined as that which takes the objective to its extrema:

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

The best possible objective is then:

$$\text{obj}^* = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

This equation measures the goodness of a tree in terms of the gain and complexity, providing a way to evaluate the performance of each tree.

Finally, for each split in the  $t$ -th boosting iteration, the gain from that split is computed as:

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

where  $G_L$  and  $H_L$  represent the summed gradients and Hessians for the left child node, and  $G_R$  and  $H_R$  represent those for the right child node. The parameters  $\lambda$  and  $\gamma$  are regularization constants. This is simply the change in goodness of a tree on adding an additional split to a leaf.

At each boosting iteration, the algorithm uses a greedy search by evaluating the gain for all possible splits across the features and selecting the split that maximizes this gain. This means that the model needs access to all the feature values in the dataset and the corresponding labels to calculate the gradients and Hessians and find the best split. This is not possible in a federated setting. The next section describes an approach that will make the greedy search work in a federated setting.

### 3.1.6 FedXGBoost

In FedXGBoost, each node has pre-defined feature splits, splitting the feature space into distinct regions. For each region, it calculates the gradients and Hessians. This means that each region has the aggregated gradient value of all the data samples that fall in it. This forms gradient and Hessian tensors that it sends to the aggregator. The aggregator uses this to build another boosting tree that it sends back to the nodes, which use it to predict once again and calculate new tensors. This is a single boosting round. This process is shown in the pseudo-code below:

---

#### Algorithm 1 Federated XGBoost Pseudocode Server Side

---

```

1: for  $c \in \mathbf{clients}$  do
2:    $\text{quantiles}[c] \leftarrow \text{get\_quantiles}(c)$ 
3: end for
4:  $\text{Quantiles} \leftarrow \mathbf{aggregate}(\text{quantiles})$ 
5:  $\text{assign\_splits}(\text{Quantiles})$ 
6: for  $n \in \mathbf{boosting\_rounds}$  do
7:    $\text{tensors} \leftarrow \text{collect\_tensors}(\mathbf{clients})$ 
8:    $\text{estimator} \leftarrow \text{train\_estimator}(\text{tensors})$ 
9:    $\text{send\_estimator}(\text{estimator}, \mathbf{clients})$ 
10: end for

```

---

---

**Algorithm 2** Federated XGBoost Pseudocode Client Side

---

```
1: while connection_active() do
2:   if quantiles_requested() then
3:     send_quantiles(server)
4:   end if
5:   if estimator_received() then
6:     estimators.append(estimator)
7:      $y_{\text{pred}} \leftarrow \text{predict}(\text{estimators}, X)$ 
8:     grads  $\leftarrow \text{compute\_grads}(y_{\text{pred}})$ 
9:     hess  $\leftarrow \text{compute\_hess}(y_{\text{pred}})$ 
10:    send_tensors(grads, hess, server)
11:   end if
12: end while
```

---

**Why Does This Approach Work for Federated Learning?** One of the main challenges in federated learning is the non-IID (Independent and Identically Distributed) nature of the data across different nodes. In FedXGBoost, data is not collected from individual nodes. Instead, each node sends only the gradients and Hessians over pre-defined regions of the feature space to the aggregator. This approach helps maintain the privacy of individual node data while still allowing the model to learn from the global distribution of the data.

By aggregating the gradients and Hessians from multiple nodes, FedXGBoost provides a global understanding of the feature distributions, which helps address the non-IID problem effectively. This aggregation reveals patterns in the global data distribution, enabling the algorithm to make informed decisions when constructing the tree and assigning splits.

**Obtaining Possible Feature Splits** To obtain possible feature splits, quantiles of the feature values are computed at each node. These quantiles are then aggregated from all nodes to form a global set of candidate split points. A set number of split points are selected from these aggregated quantiles. This method ensures that there are more splits where the feature values are more densely distributed, allowing the algorithm to make finer decisions in those regions.

**Gain-Based Assignment of Splits per Feature** In FedXGBoost, a limited number of total splits are distributed across the features. Instead of evenly distributing the splits, the splits are assigned based on the importance of each feature. In the initial training rounds, splits are divided evenly across features. As the training progresses, the total gain for each feature is calculated, and the splits are reassigned proportionally to the gain values of the features. This ensures that more important features receive a larger number of splits, leading to better performance.

**Time Complexity of FedXGBoost** Let  $s_i = 1 + \text{number of splits of the } i\text{-th feature}$ . The geometric mean of  $s_i$  is defined as:

$$s = \left( \prod_{i=1}^n s_i \right)^{\frac{1}{n}}$$

The size of the gradient and Hessian tensors is thus  $s^n$ , where  $n$  is the number of features. Given  $m$  samples per node, the region computation at each node has a time complexity of  $O(m \cdot s^n)$ .

The overall time complexity of FedXGBoost for training with  $r$  boosting rounds and a maximum tree depth  $d$  is:

$$O(r \cdot n \cdot s^n \cdot d)$$

This exponential time complexity can be a limitation, especially when the number of features and boosting rounds is large.

**Space Complexity of FedXGBoost** The space complexity of FedXGBoost is driven by the size of the gradient and Hessian tensors. Since the size of the tensors is  $s^n$ , the space complexity of the algorithm is:

$$O(k \cdot s^n)$$

where  $k$  is the total number of nodes in the federated system. This means that the memory requirements increase with the number of features and the number of nodes, which can be a concern for large-scale federated systems.

**Optimizing FedXGBoost** To mitigate the high time and space complexities, a solution is to sample features at each boosting round. This reduces the number of features considered at each round and, consequently, the size of the gradient and Hessian tensors. To compensate for the loss in accuracy due to fewer features, the number of boosting rounds can be increased.

Suppose  $\log_s m$  features are sampled per round, where  $m$  is the total number of data samples across all nodes. The time complexity now becomes:

$$O(r \cdot n \cdot s^n \cdot d) = O(r \cdot n \cdot s^{\log_s m} \cdot d) = O(r \cdot n \cdot m \cdot d)$$

Additionally, the time complexity for computing regions at each node becomes  $O(m^2)$ , which is more manageable than the previous exponential complexity.

**Feature Sampling Per Boosting Round** In each boosting round, the gain values for all features are calculated and normalized to form a probability distribution. Features are then sampled from this probability distribution, ensuring that features with higher gains are more likely to be selected. This allows FedXGBoost to focus more on the important features, while less informative features are sampled less frequently. This process improves the efficiency of the model and helps reduce the computational burden without sacrificing performance.

By adopting feature sampling and compensating for lost accuracy through additional boosting rounds, FedXGBoost achieves a more scalable and efficient federated learning approach for tree-based models.

### 3.2. Phase 2: Simulating a Federated Environment

In Phase 2, a federated learning environment was simulated using 6 local servers to mimic a real-world federated setup. This setup enabled testing of the effectiveness of the

FedXGBoost model in a decentralized context where data is distributed across multiple nodes.

**Node Setup and Communication** To initiate the federated learning process, a package was sent to each node. This package contained the following components:

- **Python environment with required libraries:** Each node received a pre-configured Python environment with all necessary dependencies and libraries installed, ensuring that all nodes could run the training without issues. This environment was standardized to avoid discrepancies between nodes.
- **API with an interface for each of the 3 model training approaches:** The package included an API that exposed an interface for each of the three model training approaches—Weighted Ensemble, Majority Vote Aggregation, and Cyclic XGBoost. This API allowed the nodes to interact with the models in a uniform way, enabling them to initiate training, send updates, and receive information from the central aggregator.
- **Setup files:** Setup files were included to configure and initialize the environment. These files ensured that the nodes were ready for training once the environment was set up.

The package also configured endpoints that were accessible by the central aggregator, which facilitated the aggregation of model updates, gradients, and Hessians sent from the individual nodes.

**Data Distribution Across Nodes** The data used for training was divided across the 6 nodes based on state-wise distributions. States with the majority of the data were chosen and split among the nodes. This ensured that each node had a substantial portion of the dataset and allowed the simulation to mirror the real-world distribution of data in federated learning, where different nodes often have access to different subsets of data.

**Handling Node Failures** One of the key aspects of federated learning is its robustness to node failures. In our simulation, if a node disconnects or becomes unavailable during the training process, the system is designed to handle such failures gracefully. The aggregator detects the failure and continues the training process with the remaining active nodes. This fault-tolerant mechanism ensures that the federated learning process is not disrupted by intermittent or temporary failures of individual nodes.

**Training Continuation and Aggregation** After the setup was complete and the data was distributed across the nodes, the training proceeded with each node running its local computations. Gradients, Hessians, and model updates were periodically sent to the central aggregator, which then performed the necessary aggregation steps. The updated models were sent back to the nodes for further training. This cycle continued until convergence or a pre-defined stopping criterion was met.

By simulating this federated environment, the scalability and robustness of the FedXGBoost model were tested, its performance in a distributed setting was assessed, and the system’s handling of issues such as data heterogeneity and node failures was evaluated.

### 3.3. Phase 3: Secure Aggregation and Communication

In Phase 3, the security aspects of federated learning were addressed, focusing on ensuring that communication between nodes and the central aggregator remains secure and that the aggregation process does not expose sensitive data. This phase covers both secure communication protocols and secure aggregation techniques to protect the integrity of the model training process.

**Secure Connection** The first step in ensuring secure communication in our federated learning setup was the establishment of a secure connection between the central aggregator and the nodes. The primary goal was to protect the communication channel from unauthorized entities that might attempt to intercept or tamper with the data being exchanged between nodes and the aggregator.

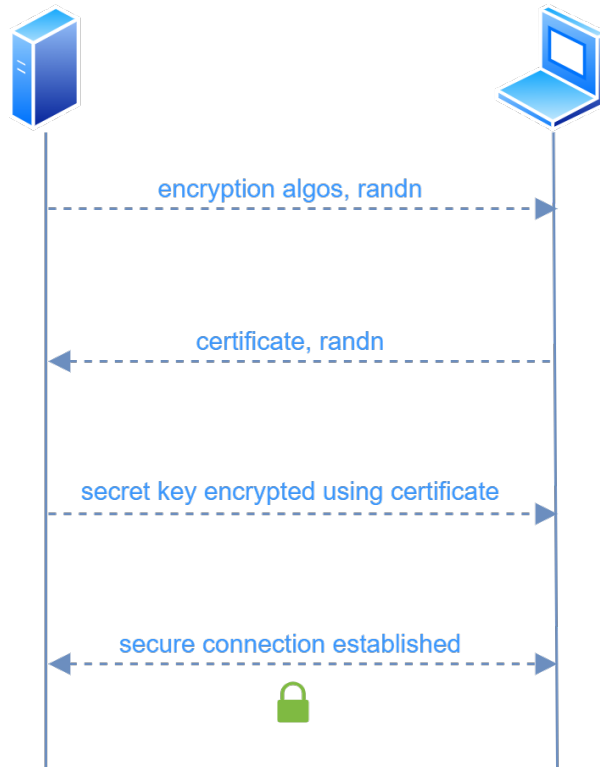


Figure 1: SSL/TLS handshake

- **SSL/TLS Encryption:** To secure the communication, SSL/TLS encryption was utilized. SSL (Secure Sockets Layer) and TLS (Transport Layer Security) are cryptographic protocols designed to provide secure communication over a network. SSL/TLS ensures that all data transmitted between nodes and the aggregator is encrypted, preventing eavesdropping, tampering, and forgery of messages. The aggregator initiates communication with the node and sends encryption algorithms along with a random number. The node returns a certificate that the aggregator verifies. The certificate also contains a public key. The aggregator generates a secret key, encrypts it using the public key, and sends it back to the node. This encrypted secret key can only be decrypted by the node's private key. Thus, the node receives a secret key, ensuring a secure connection between the node and the aggregator.

- **Self-signed certificates and keys:** Since this was still in the development phase, self-signed certificates and keys were generated and assigned to each node. These certificates served as a way to authenticate the identity of each node and the aggregator while avoiding reliance on external certificate authorities.

The use of SSL/TLS encryption with self-signed certificates ensured that all communications were protected and that only authorized nodes could participate in the federated learning process.

**Secure Aggregation** While secure communication ensured that the data exchanged between nodes and the aggregator was protected, it did not fully address the potential threat posed by a malicious aggregator. The aggregator has access to raw gradient values received from the nodes, which could potentially be used to launch reconstruction attacks. To mitigate this risk, a secure aggregation mechanism was implemented to protect the privacy of the data being shared.

**Need for Secure Aggregation** Encryption between nodes and the aggregator ensures secure communication, but it does not prevent a malicious aggregator from accessing the raw gradients or changes in model weights that are sent by the nodes. This creates the potential for *reconstruction attacks*, where an adversary with access to gradients or model weights could attempt to reconstruct the original training data.

- **Reconstruction Attack:** In a reconstruction attack, knowledge of the gradients or updates in model weights can be used to infer information about the data that was used to generate those updates. Techniques like Generative Adversarial Networks (GANs) can be employed to reproduce data that contains similarities to the original training data, posing a significant privacy risk.

**Mask-based Secure Aggregation of Gradient and Hessian Tensors** To prevent the aggregator from gaining access to sensitive information through the gradients and Hessians, a mask-based secure aggregation scheme was implemented. In this approach, each node generates a random mask and uses it to obscure its own gradient and Hessian values before sending them to the aggregator. The process is as follows:

- **Initialization:** The aggregator starts by sending an initial delta, denoted as  $D_0 = 0$ , to the first node.
- **Mask Generation:** Each node generates a random mask,  $M_i$ , and subtracts it from the delta it received. The updated delta,  $D_i$ , is then forwarded to the next node:

$$D_i = D_{i-1} - M_i$$

- **Delta Propagation:** This process continues as each node forwards the updated delta to the next node in the sequence. The final node sends its delta back to the first node, where it is added to the first node's mask:

$$M_1 = M_1 + D_n$$

This ensures that the sum of all masks is zero, providing a secure aggregation mechanism where the final result is correct, but the individual nodes' gradients and Hessians are obscured.

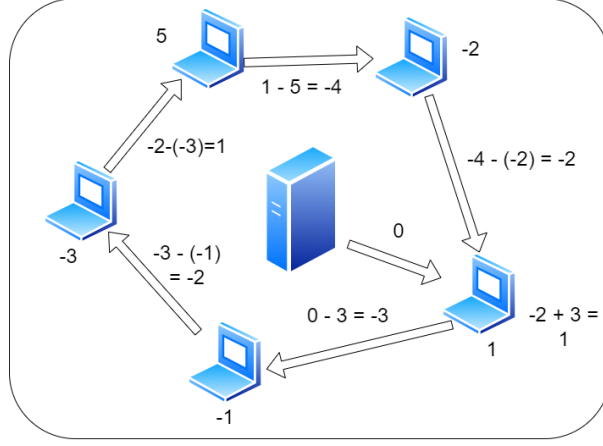


Figure 2: Masked aggregation.

**Proof that the masks sum to zero:** At each step, the delta is updated by subtracting a mask. If this process is continued for all  $n$  nodes, the final delta  $D_n$  will be the sum of all the masks subtracted from the initial delta:

$$D_1 = D_0 - M_1$$

$$D_2 = D_0 - M_1 - M_2$$

$$\vdots$$

$$D_n = D_0 - M_1 - M_2 - \dots - M_n$$

Since  $D_0 = 0$ , the final delta equation becomes:

$$(D_n + M_1) + M_2 + \dots + M_n = D_0 = 0$$

Thus, the sum of the masks is zero, ensuring that the aggregated result is accurate while the individual node data remains private.

**Protection of Node Data** Each node only knows its own mask, ensuring that no information about the node's data can be inferred from the aggregated tensors. The mask-based secure aggregation mechanism guarantees that even if the aggregator is malicious, it cannot access sensitive data from individual nodes. In addition, each node besides the last one randomly chooses the next node in the sequence. Thus, there is no way to reproduce the masks even if all the delta values are known. The mask of the first client is also updated, otherwise the delta that the second client receives would be the negative mask of the first client.

**Handling Node Failure During Training** In the event that a node disconnects or fails during training, the system recalculates the masks to ensure the integrity of the aggregation process is maintained. This ensures that the secure aggregation mechanism remains robust, even in the presence of node failures.

The mask-based secure aggregation of gradients and Hessians provides a powerful method for protecting the privacy of federated learning participants. By ensuring that the masks

cancel out during aggregation, the approach prevents the malicious aggregator from extracting any sensitive information from the nodes while still allowing for accurate model updates to be shared. This mechanism is crucial for maintaining the privacy and security of the federated learning process.

## 4. Results

### 4.1. Testing the Algorithm

To assess the performance of the FedXGBoost algorithm in a Non-IID setting, a synthetic dataset was created to simulate the real-world data distribution challenges often encountered in federated learning. The steps involved in the testing process are outlined below:

- **Dataset Generation:** A dummy dataset was constructed with multiple features, designed to simulate a real-world scenario with significant variability across different nodes.
- **Sorting the Dataset:** To introduce a Non-IID (Non-Independent and Identically Distributed) challenge, the dataset was sorted based on the values of a single feature. This sorting introduced skewed data distributions across different nodes, making it more realistic for federated learning environments, where data is typically distributed in a non-identical fashion.
- **Data Distribution Across Nodes:** After sorting, the dataset was split at different cutoff values of the sorted feature and distributed among the six nodes. This ensured that each node received data representing a distinct region of the feature space, further emphasizing the Non-IID challenge.

**Results:**

Model	Accuracy
Cyclic Xgboost	0.75
<b>FedXGBoost</b>	<b>0.93</b>
<b>Centralized XGBoost</b>	<b>0.95</b>

Table 1: Testing on Non IID dummy data

The algorithm was successfully evaluated under Non-IID conditions, and the results demonstrate that FedXGBoost effectively addresses this challenge. Its performance is comparable to that of centralized XGBoost. While cyclic XGBoost performs well on other datasets, the disparity becomes more evident on this dataset, highlighting its limitations in handling Non-IID data. In contrast, FedXGBoost excels under these conditions, delivering robust and reliable results.

### 4.2. Results on Disease Prognosis

In addition to testing the algorithm in a synthetic setting, the performance of FedXGBoost was also evaluated on a real-world dataset related to disease prognosis. Specifically, a diabetes dataset was used to assess how the model performs in predicting the progression of the disease across different federated nodes.

- **Dataset Description:** The diabetes dataset includes multiple features related to patient health and medical history, with the goal of predicting the risk of diabetes or related health issues.
- **Best Results from Each Approach:** The results from each of the three model training approaches—weighted ensemble of tree-based models, majority vote-based aggregation of random forest models, and cyclic XGBoost—were compared against the results obtained using FedXGBoost.
- **Comparison of Model Performance:** FedXGBoost outperformed the other approaches in terms of both accuracy and robustness, particularly in the presence of Non-IID data distributions. The model showed better generalization to new data, indicating its ability to effectively handle the variations introduced by federated learning settings.

The following tables and graphs summarize the best results obtained for each approach:

Model	Accuracy	F1 Score
Weighted Ensemble	0.76	0.73
Cyclic XGBoost	0.78	0.76
<b>FedXGBoost</b>	<b>0.79</b>	<b>0.79</b>
<b>Centralized</b>	<b>0.80</b>	<b>0.78</b>

Table 2: Model performance on the diabetes test set

The results demonstrate that FedXGBoost consistently achieved the highest accuracy and F1 score among all tested models. Notably, while the centralized model exhibited marginally better accuracy, FedXGBoost slightly outperformed the centralized model in terms of F1 score. These findings confirm the effectiveness of FedXGBoost’s approach to handling Non-IID data and securely aggregating model updates, resulting in superior performance for disease prognosis tasks.

## 5. Conclusion

The results from both synthetic testing and real-world disease prognosis applications demonstrate that FedXGBoost is a robust and efficient algorithm for federated learning. It addresses the challenges posed by Non-IID data, improves the performance of tree-based models in federated settings, and ensures secure aggregation of model updates. The algorithm’s ability to perform well in a variety of scenarios highlights its potential for widespread use in federated learning tasks, particularly in healthcare and other domains requiring privacy-preserving model training. The sampling strategy dials down the time complexity of the algorithm whilst ensuring that good features are sampled more often, substantially improving the greedy search approach. Masked aggregation coupled with secure communication maintain the privacy of client data and ensure a robust defense against reconstruction attacks. The presented framework thus demonstrated its practicality and scalability in real world federated settings.

## References

- [1] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, Blaise Aguera y Arcas. *Communication-Efficient Learning of Deep Networks from Decentralized Data*. 2017.
- [2] Hong-You Chen, Wei-Lun Chao. *FEDBE: Making Bayesian Model Ensemble Applicable to Federated Learning*. 2021.
- [3] Yuya Jeremy Ong, Yi Zhou, Nathalie Baracaldo, Heiko Ludwig. *Adaptive Histogram-Based Gradient Boosted Trees for Federated Learning*. IBM Research - Almaden. 2021.
- [4] William Lindskog, Christian Prehofer, Sarandeep Singh. *Histogram-Based Federated XGBoost using Minimal Variance Sampling for Federated Tabular Data*. 2021.
- [5] Ligeng Zhu, Zhijian Liu, Song Han. *Deep Leakage from Gradients*. CoRR abs/1906.08935 (2019). <http://arxiv.org/abs/1906.08935>
- [6] Briland Hitaj, Giuseppe Ateniese, Fernando Perez-Cruz. *Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning*. CCS '17, Association for Computing Machinery, 2017.
- [7] Nicholas Carlini, et al. *Extracting Training Data from Large Language Models*. 2021.

### A. Class Diagram A

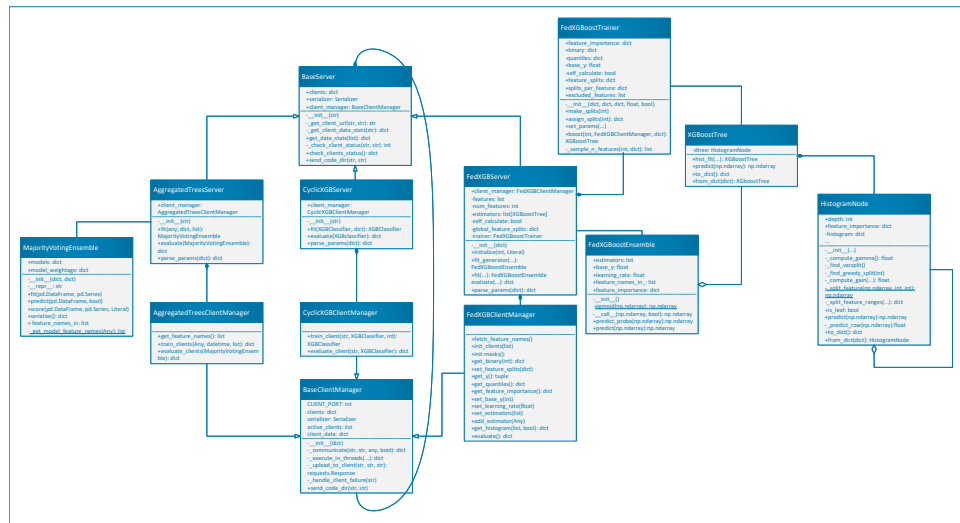


Figure 3: Aggregator Class Diagram

## B. Class Diagram B

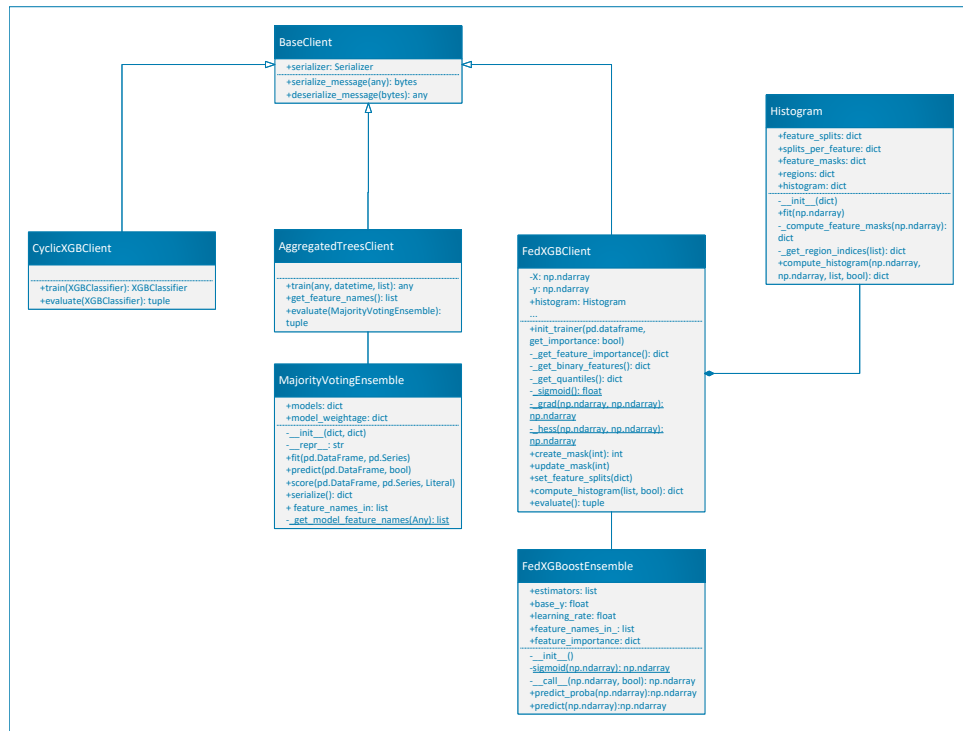


Figure 4: Nodes Class Diagram