# Explainable Federated Learning by Incremental Decision Trees ⋆

Parisa Jamshidi[1,*], Sławomir Nowaczyk[1], Mahmoud Rahat[1] and
Zahra Taghiyarrenani[1]

[1]Center for Applied Intelligent Systems Research (CAISR),
Halmstad University, Halmstad, Sweden

**Abstract**

Explainable Artificial Intelligence (XAI) is crucial in ensuring transparency, accountability, and trust in machine learning models, especially in applications involving high-stakes decision-making. This paper focuses on addressing the research gap in federated learning (FL), specifically emphasizing the use of inherently interpretable underlying models. While most FL frameworks rely on complex, black-box models such as Artificial Neural Networks (ANNs), we propose using Decision Tree (DT) classifiers to maintain explainability. More specifically, we introduce a novel framework for horizontal federated learning using Extremely Fast Decision Trees (EFDTs) with streaming data on the client side. Our approach involves aggregating clients' EFDTs on the server side without centralizing raw data, and the training process occurs on the clients' sides. We outline three aggregation strategies and demonstrate that our methods outperform local models and achieve performance levels close to centralized models while retaining inherent explainability.

**Keywords**

eXplainable AI (XAI), Federated Learning, Incremental Decision Tree, Extremely Fast Decision Tree, Data Stream

## 1. Introduction

Explainable Artificial Intelligence (XAI) is essential in machine learning and artificial intelligence as it improves transparency, accountability, trust, and the ability to enhance and debug AI systems. As AI technologies expand to various sectors, from healthcare to finance, the complexity and opacity of these models often prevent their deployment in critical decision-making processes. XAI seeks to mitigate this issue by providing insights into how AI algorithms reason and how they arrive at specific conclusions, enabling users to validate and comprehend model behaviors. Most of XAI research today focuses on post-hoc explanations for black-box models. These models have complex internal workings that are difficult for humans to interpret or understand. Various forms of analysis, from surrogate models to gradient credit assignment, are often used to explain these models and their decisions. However, these post-hoc explanations have several drawbacks. For example, they often fail to capture the true inner workings of the model,

*TempXAI@ECML-PKDD'24: Explainable AI for Time Series and Data Streams Tutorial-Workshop, Sep. 9th, 2024, Vilnius, Lithuania*

✉ parisa.jamshidi@hh.se (P. Jamshidi); slawomir.nowaczyk@hh.se (S. Nowaczyk); mahmoud.rahat@hh.se (M. Rahat); zahra.taghiyarrenani@hh.se (Z. Taghiyarrenani)

iD 0000-0001-7055-2706 (P. Jamshidi); 0000-0002-7796-5201 (S. Nowaczyk); 0000-0003-2590-6661 (M. Rahat); 0000-0002-1759-8593 (Z. Taghiyarrenani)

providing only surface-level insights that might not be trustworthy [1, 2]. Furthermore, they are vulnerable to adversarial attacks that can manipulate the explanations without changing the model's predictions [3, 4]. Such limitations make them a questionable choice in applications where transparency is a priority. Conversely, white-box models are inherently explainable models designed to be transparent and interpretable from the outset. These models allow users to understand their decision-making processes without the need for additional interpretability techniques [1].

Federated learning (FL) is a technique in machine learning in which multiple devices record local data and share their model with a server. The server forms a global model and shares it without exchanging raw data. This method is more efficient regarding bandwidth and computing resources because it does not need to transfer large amounts of raw data to a central server. Although this approach has gained much attention due to its capacity, most of the underlying models for FL are Artificial Neural Networks (ANNs). Post-hoc explainability methods can be applied to black-box models of FL. For example, [5] used feature importance methods to add explainability to their models, while [6] presented counterfactual explanation techniques. However, there is a surprising lack of FL techniques suitable for inherently interpretable models.

In this paper, we focus on Decision Tree (DT) classifiers. Shallow DTs, in contrast to NNs and complex models, are designed in a way that their internal workings are easily understandable to humans. This allows for straightforward tracing of how inputs are transformed into outputs, facilitating immediate insight into the decision-making process. Such transparency is crucial for validating the model's logic, ensuring ethical compliance, and fostering trust among users. This makes DTs particularly suitable for the identification and correction of biases and errors within the model: their transparent nature enables users to pinpoint specific aspects of the model that contribute to undesirable outcomes, facilitating targeted improvements. Moreover, some research indicates that tree-based models outperform ANNs in some applications, specifically tabular data [7, 8].

The typical approach in ML is batch learning, also known as offline learning, which involves training a model on a fixed dataset all at once. On the other hand, online learning, or incremental learning, continuously updates the model as new data becomes available. This makes it suitable for dynamic environments where data arrives sequentially. Much less research has been done with incremental learning in FL fashion than batch learning.

Therefore, there is a gap when clients have access to streaming data and aim to achieve an inherently explainable global model with better performance than their local model through collaboration in a federated mode. We proposed our method to aggregate the incremental decision trees from clients' sides in an FL framework, which can achieve higher performance compared to local ones, and as the client's models are DTs, they are inherently explainable. Our contributions are as follows:

- We created a global model without centralizing raw data, using statistical information stored in EFDTs.
- In our proposed method, EFDTs are trained locally and aggregated in each round.
- We have introduced three aggregation methods on the server to aggregate clients' EFDTs.
- Furthermore, we maintain both local and global models as a single tree to make them inherently explainable.

The rest of the paper is organized as follows: Related work summarizes previous research intersection of FL and tree-based methods. Then, in Preliminaries, we introduce some basic concepts of incremental decision trees and tree similarity methods. The proposed method is described in Methodology. The experimental setup is found in Section Experimental Setup followed by the result in Results. Finally, we draw some conclusions in Conclusion

## 2. Related work

Within this distributed learning paradigm, horizontal federated learning (HFL) and vertical federated learning (VFL) represent two approaches. Each client has different data samples in HFL, but the feature set is the same. VFL, however, uses data samples shared among clients with different features [9].

SecureBoost [10] is a VFL framework in which only one of the clients has the label of samples, called active party, and the rest, without labels, called passive parties. This method works with a gradient-tree boosting algorithm in which the active party shares the gradient and Hessian values with the passive party. After that, the passive party categorizes the samples based on local features into buckets and provides the total values of each bucket to the active party in the form of a histogram. The active party then performs local calculations to find the best way to split the nodes and directly coordinate updates with the passive parties. Other methods, like OpBoost [11], were proposed to optimize SecureBoost. There are other methods categorized on tree-based VFL, including FEDXGB [12] and VF2Boost [13], which are Homomor Encryption (HE)-based.

There is also research on the conjugation of tree-based models and HFL. [14] proposed a method based on GBDT(Gradient-boosted decision trees), in which each client trains a decision tree using their local dataset and adds it to the global model in turn. Boosting-based Federated Random Forest (BOFRF) [15] proposed a boosting framework for random forests in a federated manner. Each client forms their RFs and, on the server side, considers those as weak classifiers and proposes a method to calculate their weights. Our approach involves explicitly training a single tree, which is explainable, whereas using the bagging approach in the server to form the global model will result in a loss of explainability.

Another approach is to train a tree-based model collaboratively in a server. Clients share local statistical information with the server, and the server decides on node partitioning in each round.

There are few works of conjugation of incremental decision trees and FL. [16] and [17] are two methods to train an IDT in VFL and HFL, respectively. The first method encrypts samples from clients' streaming data and passes them to the server. In the latter method, the Data Collector collected the raw data from the client and aggregated them. In both, the server uses that information to train a VFDT.

Our proposed method falls under the category of HFL, and the features are consistent across all clients. Additionally, we do not share raw data with the server. Instead, we train trees on the client side, and the server is responsible for aggregating the trees.

# 3. Preliminaries

## 3.1. Incremental Decision tree

The Hoeffding Tree (HT) [18] is one of the fundamental research studies that proposed the Incremental Decision Tree (IDT) for efficient data stream mining. The HT uses the Hoeffding bound to determine when a statistically significant decision about splitting a node can be made based on the amount of data collected. The inequality for a random variable $r$ with range $R$ is expressed as follows:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}, \tag{1}$$

where $\epsilon$ represents the margin of error, $\delta$ is the confidence level, and $n$ is the number of observations.

In 2018, [19] proposed the Hoeffding Anytime Tree, also known as the Extremely Fast Decision Tree (EFDT). EFDT, in contrast, enhances the capabilities of HT by including an ”anytime” aspect. This means it can continuously update and revise its model as new data becomes available without revisiting old data. While HT makes decisions based on accumulated data up to a certain point, EFDT continuously improves its decisions, resulting in better performance in dynamic data streams.

## 3.2. Decision Tree Similarity

An important step in Federated Learning is an aggregation of models from different clients, and a crucial component in that process is based on the similarity between them. Decision tree similarity methods aim to quantify the similarity between different decision trees. Over the years, several approaches have been proposed, typically divided into two main categories: syntactic and semantic similarities.

**Syntactic similarity** methods compare and analyze decision trees based on their structure. Techniques such as tree edit distance, which calculates the minimal changes needed to transform one tree into another, and structural similarity measures, which compare the arrangement and features of nodes, are commonly used [20, 21, 22].

**Semantic similarity** methods aim to capture the functional similarity between trees by considering the decisions made by each tree, which means they look at the similarity in the prediction distributions [23, 22]. This can consider the similarity of decisions made by each tree on a set of instances.

# 4. Methodology

Our methodology consists of two phases: training and aggregation. Training occurs on the clients' sides, while aggregation occurs on the server's side. Our training process is incremental since we are in the streaming setting. Our main contribution resides in the aggregation phase, in which we put different aggregation strategies into contrast.

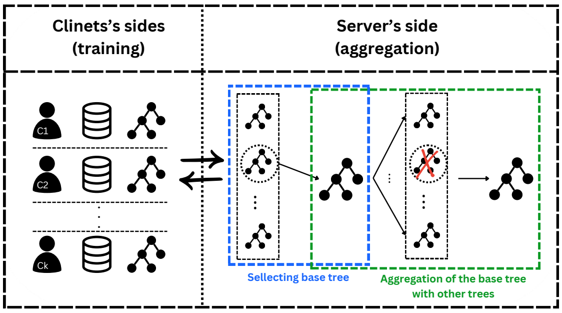

**Figure 1:** The framework of the proposed method. At the clients' sides, training takes place. The blue dashed box signifies the selection of the base tree, while the green dashed box represents the combination of the base tree with other trees, all located on the server side. Depending on the chosen strategy, the base tree may consist of one or multiple trees.

### 4.1. Problem Formulation

Assume we have $k$ clients $c_1 \cdots c_k$ where each one has access to a different streaming dataset $D_1 \cdots D_k$. Given that we are in a horizontal federated setting, the feature set is the same for all datasets $f_1 \cdots f_m$.

In particular, in order to maintain the inherent interpretability of the global model, we require that the final result be limited to a single decision tree. While aggregating a number of trees into a forest is a natural way to extend DT learning to a federated setting, our work is one of the very few attempts to collaboratively create a single tree among all the clients.

In our proposed method, the server only aggregates the clients' trees without training them, so there's no need to share raw data with the server.

### 4.2. Incremental training on the client side

Each client($c_i$) trains its own decision tree on its local dataset($D_i$). As a new data point arrives, they use incremental learning to update their tree. After a predefined number of data points are received by each client, they send their trained decision trees to the server. Then, each one will receive a new tree from the server and continue using and training this new tree by the next part of their streaming data.

Each node of EFDT stores a $n_{ijk}$ table, which can be interpreted as the number of instances where the $i-th$ attribute has the $j-th$ value, and the class label is $k$. This table is used to calculate metrics such as information gain or Gini index, which determine the best attribute for splitting the data at a node. Each client will share their tree with this information instead of raw data.

### 4.3. Aggregation on the server side

The aggregation process begins when the server receives decision trees from the clients. We have proposed three different strategies for the server side. However, the aggregation process remains consistent across all strategies, but each strategy differs in its approach to selecting the base tree. Before delving into the specifics of each strategy, let's first explore how the aggregation process operates.

As mentioned earlier, every node in EFDT contains an $n_{ijk}$ table. As part of the aggregation process, we merge the tables of nodes along the same path into two trees. Algorithm 1 outlines this method in detail. Specifically, when two nodes from two trees share common parents, they can be effectively combined, even if they have opted for different features. This is possible because the table contains aggregated information before the split.

**Algorithm 1** Aggregation Function

```
function AGGREGATION($T_1$, $T_2$)
    function TRAVERSE($node1$, $node2$)
        if $node1$ is None or $node2$ is None then
            exit
        else
            merge($node1.n_{ijk}$, $node2.n_{ijk}$)
        end if
        if $node1$.feature == $node2$.feature and $node1$.value == $node2$.value then
            TRAVERSE($node1$.right_child, $node2$.right_child)
            TRAVERSE($node1$.left_child, $node2$.left_child)
        else
            exit
        end if
    end function
    TRAVERSE($T_1$.root, $T_2$.root)
end function
```

As mentioned earlier, the strategies differ in terms of how the base tree is chosen to combine the clients' models. Below are the details of three proposed strategies.

### 4.3.1. Syntactic strategy (Syn):

In this approach, the base tree is the most representative syntactic tree among the clients' trees, meaning it is the tree that requires the fewest changes to match the other trees. By using the edit distance on the bracket format of the trees, we can identify this tree, which we refer to as ”Syn” in Algorithm 2. Then, the aggregation method is applied to ”Syn” along with the other clients' trees. Finally, the server broadcasts the Syn tree to all clients.

### 4.3.2. Semantic strategy (Sem):

The base tree in this strategy is that tree with minimum disagreement with other clients on predictions. In Algorithm 3, the server uses a proxy dataset, $D_s$, to identify this tree, referred to as ”Sem”. Then, it iterates through all client trees, aggregating information from matching paths in each tree into the Sem tree to update it. Finally, the server broadcasts the Sem tree to all clients.

**Algorithm 2** Syntactic strategy (Syn)

```
Input: List of clients' trees $clientsTree = [C_1, C_2, \dots, C_k]$

Output: one global tree
$BFs \leftarrow \emptyset$        ▷ To store bracket format of $clientsTree$
for each $C_i$ in clientsTree do
    $BFs$.add(BracketFormat($C_i$))
end for
$Syn \leftarrow SelectBaseTree(BFs)$        ▷ Representative Syntactic tree
for each $C_i$ in clientsTree do
    if $C_i \neq Syn$ then
        AGGREGATION($Syn$, $C_i$)
    end if
end for
return $Syn$
```

**Algorithm 3** Semantic strategy (Sem)

```
Input: List of clients' trees $clientsTree = [C_1, C_2, \dots, C_k]$
dataset located on server: $D_s$
Output: one global tree
$Pred \leftarrow \emptyset$        ▷ To store $clientsTree$ predictions of $D_s$
for each $C_i$ in $clientsTree$ do
    $Pred$.add(prediction($C_i$, $D_s$))
end for
$Sem \leftarrow SelectBaseTree(Pred)$        ▷ Representative Semantic tree
for each $C_i$ in $clientsTree$ do
    if $C_i \neq Sem$ then
        AGGREGATION($Sem$, $C_i$)
    end if
end for
return $Sem$
```

#### 4.3.3. Individual strategy (Ind):

In contrast to previous strategies, this strategy constructs one tree per client. It compares each client's tree with the trees of all other clients and aggregates information from matching paths into the final tree for that client (Algorithm 4). After updating all trees, the server will pass the updated trees to their owners.

**Algorithm 4** Individual strategy (Ind)

```
Input: List of clients' trees $clientsTree = [C_1, C_2, \dots, C_k]$
Output: List of individual aggregated trees for each client
$Inds$.copy($clientsTree$)
for each $Ind$ in $Inds$ do
    for each $C_i$ in $clientsTree$ do
        if $C_i \neq Ind$ then
            AGGREGATION($Ind$, $C_i$)        ▷ Each tree would be aggregated with the other trees' common path.
        end if
    end for
end for
```

We used centralized learning and local learning to establish upper-bound and lower-bound, respectively, for the mentioned strategies. In centralized learning, all the data exist in a single location, and a single model is trained. In contrast, in local learning, each client trains a model with its own data without interacting with other clients.

# 5. Experimental Setup

## 5.1. Dataset Description

We use two datasets with categorical features, which lets us better control the experiment and analyze the results. Below is a brief description of each.

**Mushroom dataset:** This dataset contains 8124 descriptions of hypothetical samples representing 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is classified as edible or poisonous. There are 22 features and 8124 data points[1].

**Chess (King-Rook vs. King-Pawn):** This dataset contains two classes: white-can-win (”won”) and white-cannot-win (”nowin”). The classes are mostly balanced at 52% and 48%, respectively. There are 36 features and 3196 data points[2].

## 5.2. Experiments Description

To share the data across clients, we used stratified splitting so that the ratio of the classes was the same for all clients. Moreover, the amount of data each client receives is the same as other clients. To address the uncertainty, we randomly performed this data-sharing schema 10 times with different seeds, and the result was averaged.

It's important to note that before dividing the data among clients, 1% of the data is set aside to be stored on the server side. This data, referred to as $D_s$ in this paper, is used to identify the representative semantic tree.

Our model choice is Extremely Fast Decision Tree (EFDT) [19], which is the same for all clients. For stable results, we resorted to the River library in Python [24]. The parameters for EFDT are set to be the default parameters; only the binary_split is set to True. To construct the representative syntactic tree used in SynRoot and SynPath, the APTED library is utilized [20, 21].

For the federated learning setting, we consider 4 clients and 200 rounds of communication between the clients and the server. Accordingly, each client's data is divided by 200 and fed in a streaming manner to clients' EFDT. This learning procedure is repeated for each strategy mentioned in 4.

# 6. Results

In this section, we present the results obtained using the proposed method. We compare the result of the proposed method with different strategies against centralized and local learning. We present two tables, Tables 1 and 2, showing the accuracy of five strategies after some communication rounds for the mushroom and chess datasets, respectively. In almost all cases, the performance of the proposed strategies falls between the performance of local and centralized learning, as expected. These strategies benefit from aggregation, allowing them to access more information at each point compared to local learning. However, their access to information is still not as extensive as centralized learning.

[1] https://archive.ics.uci.edu/dataset/73/mushroom

[2] https://archive.ics.uci.edu/dataset/22/chess+king+rook+vs+king+pawn

**Table 1**

Mean ± std of the accuracy(%) of 10 runs for different methods in some rounds(Mushroom dataset)

| #round | Centralized | Local | Ind | Syn | Sem |
|---|---|---|---|---|---|
| **1** | 48.97 ± 9.13 | 44.44 ± 11.11 | 44.44 ± 11.11 | 44.44 ± 11.11 | 44.44 ± 11.11 |
| **5** | 70.15 ± 4.39 | 53.83 ± 2.54 | 64.29 ± 5.22 | 65.51 ± 6.25 | 64.23 ± 6.99 |
| **15** | 82.50 ± 1.81 | 66.31 ± 2.02 | 79.60 ± 1.75 | 79.90 ± 1.99 | 79.55 ± 2.37 |
| **25** | 86.38 ± 1.51 | 72.81 ± 1.63 | 83.53 ± 1.27 | 83.79 ± 1.20 | 83.45 ± 1.38 |
| **50** | 89.65 ± 1.05 | 80.58 ± 0.82 | 87.52 ± 0.84 | 87.74 ± 0.65 | 87.54 ± 0.87 |
| **100** | 92.92 ± 0.45 | 85.97 ± 0.48 | 90.66 ± 0.58 | 90.80 ± 0.52 | 90.64 ± 0.54 |
| **150** | 94.86 ± 0.48 | 88.27 ± 0.41 | 92.37 ± 0.54 | 92.44 ± 0.61 | 92.20 ± 0.54 |
| **175** | 95.51 ± 0.44 | 88.95 ± 0.46 | 92.96 ± 0.50 | 93.06 ± 0.56 | 92.80 ± 0.51 |
| **185** | 95.73 ± 0.42 | 89.22 ± 0.41 | 93.20 ± 0.54 | 93.31 ± 0.60 | 93.06 ± 0.54 |
| **195** | 95.92 ± 0.41 | 89.50 ± 0.39 | 93.42 ± 0.53 | 93.55 ± 0.58 | 93.30 ± 0.55 |
| **200** | 96.01 ± 0.40 | 89.62 ± 0.38 | 93.52 ± 0.54 | 93.66 ± 0.59 | 93.42 ± 0.56 |

**Table 2**

Mean + std of the accuracy(%) of 10 runs for different methods in some rounds(Chess dataset)

| #round | Centralized | Local | Ind | Syn | Sem |
|---|---|---|---|---|---|
| **1** | 48.18 ± 14.11 | 45.00 ± 18.71 | 45.00 ± 18.71 | 45.00 ± 18.71 | 45.00 ± 18.71 |
| **5** | 57.12 ± 4.22 | 45.54 ± 6.83 | 54.64 ± 4.87 | 56.25 ± 6.20 | 53.57 ± 4.72 |
| **15** | 63.30 ± 2.62 | 51.59 ± 4.33 | 61.08 ± 2.99 | 61.88 ± 3.51 | 59.55 ± 3.93 |
| **25** | 66.96 ± 2.44 | 56.66 ± 2.56 | 64.56 ± 3.23 | 65.74 ± 3.02 | 63.18 ± 4.06 |
| **50** | 69.30 ± 1.92 | 61.12 ± 1.23 | 66.33 ± 2.05 | 66.68 ± 2.13 | 65.72 ± 2.59 |
| **100** | 77.11 ± 2.31 | 65.49 ± 1.22 | 72.63 ± 2.55 | 72.92 ± 2.78 | 72.27 ± 2.92 |
| **150** | 81.83 ± 2.01 | 66.99 ± 1.38 | 77.34 ± 2.95 | 77.80 ± 3.03 | 77.31 ± 3.06 |
| **175** | 83.46 ± 1.87 | 67.54 ± 1.08 | 78.96 ± 3.00 | 79.49 ± 2.90 | 79.07 ± 2.91 |
| **185** | 83.97 ± 1.80 | 67.82 ± 1.01 | 79.39 ± 2.93 | 79.88 ± 2.88 | 79.49 ± 2.79 |
| **195** | 84.49 ± 1.72 | 68.14 ± 1.01 | 79.94 ± 2.87 | 80.51 ± 2.80 | 80.07 ± 2.68 |
| **200** | 84.65 ± 1.62 | 68.30 ± 0.94 | 80.14 ± 2.78 | 80.71 ± 2.71 | 80.27 ± 2.58 |

Then, we look deeper at the results on the mushroom dataset shown in Figure 2. We observe that the speed of performance improvement in the three methods presented is initially higher than that of local learning (refer to the dashed box in the lower left part of Figure 2). However, this difference decreases after several stages, eventually reaching less than 5% in the last rounds (see the dashed box in the upper right part of Figure 2). This suggests that a client with initially limited data cannot effectively compete with aggregated methods. With access to more data, the client may better understand the problem and achieve strong performance independently close to aggregated methods.

In all the experiments with the chess dataset, there are significant differences between the performance of the proposed methods and local learning performance. If you compare the dashed boxes in Figure 3, you'll notice that individual clients cannot achieve the same performance as aggregated methods even after 200 rounds. However, similar to the results of the mushroom dataset, the performance of the aggregated methods improves quickly in the initial rounds in this dataset as well.

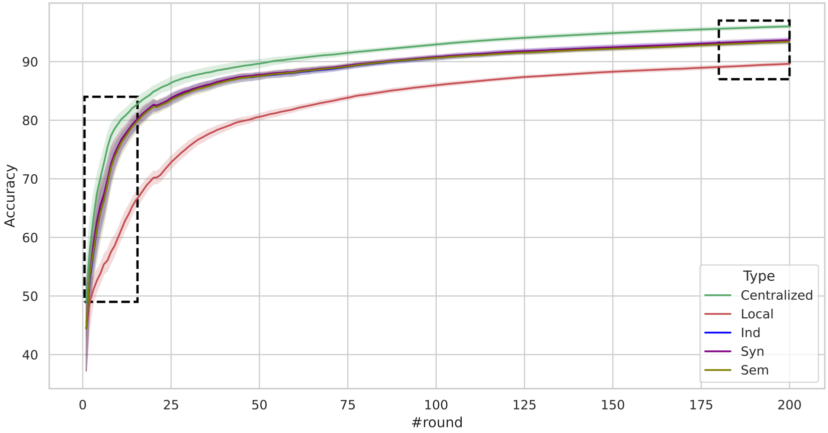

**Figure 2:** The performance of Syn, Sem, and Ind strategies, along with local and centralized learning of experiments on the mushroom dataset. The x-axis represents the number of communication rounds between clients and servers, and the y-axis represents the accuracy.

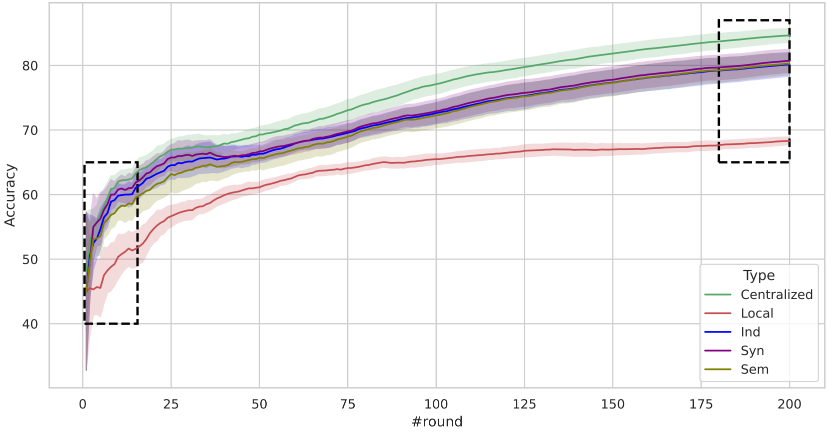

**Figure 3:** The performance of Syn, Sem, and Ind strategies, along with local and centralized learning of experiments on the chess dataset. The x-axis represents the number of communication rounds between clients and servers, and the y-axis represents the accuracy.

## 7. Conclusion

This paper introduces a framework for horizontal federated learning using an Extremely Fast Decision Tree (EFDT) as the underlying model and streaming data on the client side. We propose three strategies to aggregate this type of underlying method. With our strategies, you can merge the information on multiple EFDTs at each round, which helps to enhance the performance at the client level. We also insist on having only one tree on the client side because a single (shallow) tree is inherently explainable. We avoid using black-box models due to concerns about using post-hoc explanation methods.

We compared these three strategies with local and centralized learning. All three strategies produce similar results, and their performance is higher than local learning and close (yet low) to centralized learning. Experiments show that although, in some cases, the performance of the proposed methods might not differ too much from local learning after some rounds, they always grow very fast in the first rounds.

## Acknowledgments

The work was carried out with support from The Knowledge Foundation and from Vinnova (Sweden's innovation agency) through the Vehicle Strategic Research and Innovation Programme, FFI.

## References

[1] C. Rudin, Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, Nature machine intelligence 1 (2019) 206–215.

[2] P.-J. Kindermans, S. Hooker, J. Adebayo, M. Alber, K. T. Schütt, S. Dähne, D. Erhan, B. Kim, The (un) reliability of saliency methods, Explainable AI: Interpreting, explaining and visualizing deep learning (2019) 267–280.

[3] D. Alvarez-Melis, T. S. Jaakkola, On the robustness of interpretability methods, arXiv preprint arXiv:1806.08049 (2018).

[4] D. Slack, S. Hilgard, E. Jia, S. Singh, H. Lakkaraju, Fooling lime and shap: Adversarial attacks on post hoc explanation methods, in: Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, 2020, pp. 180–186.

[5] G. Wang, Interpret federated learning with shapley values, arXiv preprint arXiv:1905.04519 (2019).

[6] P. Chen, X. Du, Z. Lu, J. Wu, P. C. Hung, Evfl: An explainable vertical federated learning for data-oriented artificial intelligence systems, Journal of Systems Architecture 126 (2022) 102474.

[7] A. Karlsson, T. Wang, S. Nowaczyk, S. Pashami, S. Asadi, Mind the data, measuring the performance gap between tree ensembles and deep learning on tabular data, in: International Symposium on Intelligent Data Analysis, Springer, 2024, pp. 65–76.

[8] L. Grinsztajn, E. Oyallon, G. Varoquaux, Why do tree-based models still outperform deep learning on typical tabular data?, Advances in neural information processing systems 35 (2022) 507–520.

[9] Z. Wang, K. Gai, Decision tree-based federated learning: A survey, Blockchains 2 (2024) 40–60.

[10] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, Q. Yang, Secureboost: A lossless federated learning framework, IEEE Intelligent Systems 36 (2021) 87–98.

[11] X. Li, Y. Hu, W. Liu, H. Feng, L. Peng, Y. Hong, K. Ren, Z. Qin, Opboost: a vertical

federated tree boosting framework based on order-preserving desensitization, arXiv preprint arXiv:2210.01318 (2022).

[12] Y. Liu, Z. Ma, X. Liu, S. Ma, S. Nepal, R. H. Deng, K. Ren, Boosting privately: Federated extreme gradient boosting for mobile crowdsensing, in: 2020 IEEE 40th international conference on distributed computing systems (ICDCS), IEEE, 2020, pp. 1–11.

[13] F. Fu, Y. Shao, L. Yu, J. Jiang, H. Xue, Y. Tao, B. Cui, Vf2boost: Very fast vertical federated gradient boosting for cross-enterprise learning, in: Proceedings of the 2021 International Conference on Management of Data, 2021, pp. 563–576.

[14] L. Zhao, L. Ni, S. Hu, Y. Chen, P. Zhou, F. Xiao, L. Wu, Inprivate digging: Enabling tree-based distributed data mining with differential privacy, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, 2018, pp. 2087–2095.

[15] M. Gencturk, A. A. Sinaci, N. K. Cicekli, Bofrf: A novel boosting-based federated random forest algorithm on horizontally partitioned data, IEEE Access 10 (2022) 89835–89851.

[16] Z. Han, C. Ge, B. Wu, Z. Liu, Lightweight privacy-preserving federated incremental decision trees, IEEE Transactions on Services Computing (2022).

[17] S. Sharma, K. Arora, P. S. Thakur, Horizontal federating decision tree learning from data streams: Building intelligence in iot edge networks, in: 2022 IEEE 8th World Forum on Internet of Things (WF-IoT), IEEE, 2022, pp. 1–6.

[18] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, 2000, pp. 71–80.

[19] C. Manapragada, G. I. Webb, M. Salehi, Extremely fast decision tree, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 1953–1962.

[20] M. Pawlik, N. Augsten, Efficient computation of the tree edit distance, ACM Transactions on Database Systems (TODS) 40 (2015) 1–40.

[21] M. Pawlik, N. Augsten, Tree edit distance: Robust and memory-efficient, Information Systems 56 (2016) 157–173.

[22] A. I. Weinberg, M. Last, Selecting a representative decision tree from an ensemble of decision-tree models for fast big data classification, Journal of Big Data 6 (2019) 1–17.

[23] R. Miglio, G. Soffritti, The comparison between classification trees through proximity measures, Computational statistics & data analysis 45 (2004) 577–593.

[24] J. Montiel, M. Halford, S. M. Mastelini, G. Bolmier, R. Sourty, R. Vaysse, A. Zouitine, H. M. Gomes, J. Read, T. Abdessalem, et al., River: machine learning for streaming data in python (2021).