Towards Neuro-Symbolic Federated Tree Learning

Andrea Augello
Department of Engineering
University of Palermo
Palermo, Italy
andrea.augello01@unipa.it

Stefano Merendino
Department of Engineering
University of Palermo
Palermo, Italy
stefano.merendino@unipa.it

Alessandra De Paola
Department of Engineering
University of Palermo
Palermo, Italy
alessandra.depaola@unipa.it

Giuseppe Lo Re
Department of Engineering
University of Palermo
Palermo, Italy
giuseppe.lore@unipa.it

***Abstract*—Federated Learning (FL) is a distributed machine learning paradigm that enables multiple clients to collaboratively train a model while keeping their data localized, thus preserving privacy. While most FL approaches focus on deep learning models, there is a growing interest in exploring alternative models that can offer interpretability and efficiency. This paper presents TreeClimber, a novel neuro-symbolic approach for training decision tree models in a federated setting using gradient-free optimization techniques. TreeClimber embeds data into a subsymbolic latent space via a randomized embedding, from which decision trees are reconstructed through a neural architecture. Trees are optimized in the embedding space in a federated setting using a zeroth-order approximation of the gradient, allowing for effective training of interpretable decision models without direct access to data. Experimental results demonstrate the effectiveness of TreeClimber in enabling secure distributed training of decision trees without sharing raw information about local datasets, achieving competitive performance compared to traditional centralized training methods.**

***Index Terms*—Decision Trees, Federated Learning, Gradient-free Optimization, Neuro-symbolic AI**

I. Introduction

In complex distributed systems, making reliable and interpretable data-driven decisions while preserving data privacy is a critical challenge. Federated Learning (FL) [1] has emerged as a powerful paradigm for cooperatively training machine learning models across multiple decentralized devices or servers while preserving data privacy [2]. FL has been widely adopted in various applications where data privacy is of paramount importance, such as human activity recognition [3], healthcare [4], and cybersecurity [5]. However, most FL approaches primarily focus on black-box deep learning models. These models often lack interpretability, which is a key requirement in domains where understanding the decision-making process is essential [6]. Additionally, deep learning models can be computationally intensive, making them less suitable for resource-constrained and edge devices. These challenges are particularly relevant in distributed edge/fog intelligent systems, where privacy preservation and computational efficiency are critical requirements [7]–[9].

Decision trees are widely used machine learning models known for their interpretability and efficiency, and typically outperform deep learning models on tabular data [10], making them the best choice for many real-world applications. However, while FL has been extensively studied for deep learning models, its application to decision tree-based models is still in its infancy, especially in the horizontal FL setting, where clients share the same feature space but have different samples. Existing approaches for federated decision tree learning either rely on data sharing through complex cryptographic techniques, which are unsuitable for resource-constrained devices [11], or rely on heuristic methods that may not guarantee optimal model performance and only marginally follow the FL paradigm.

Khraisat et al. [12] proposed a federated random forest approach where each client trains a local random forest, and the server combines these models to create a global model. While this method preserves data privacy, it may not fully leverage the potential of collaborative learning, as each client operates independently, and the global model grows in complexity with the number of clients, losing both interpretability and efficiency. Lim and Park. [13] propose an approach in which clients take turns training a shared decision tree model by sequentially adding nodes based on their local data. This method might lead to suboptimal tree structures, especially when the clients have heterogeneous datasets, as model growth is heavily influenced by the order of client participation and local data distributions.

To address these challenges in federated tree learning, this paper introduces TreeClimber, a novel neuro-symbolic approach for federated decision tree learning that can effectively train a unified tree model across multiple clients while preserving data privacy through a neural encoding/decoding of tree structures. TreeClimber leverages a randomized Weisfeiler-Lehman (WL) embedding [14] and a recurrent neural network to map trees into a subsymbolic latent space and reconstruct them back. Clients can collaboratively optimize the tree embeddings using a gradient-free optimization technique based on a zeroth-order approximation of the gradient [15]. The pseudo-gradient-based optimization allows for effective training that directly follows the original FL paradigm, and can condense the knowledge from all clients into a single interpretable decision tree model. It is worth noting that the TreeClimber pipeline is dataset-agnostic, meaning that no special fine-tuning is required to adapt its structure to different datasets, and TreeClimber can be easily integrated in any out-of-the-box federated learning scenario. While TreeClimber is not the first work to explore vectorial representations of decision trees [16], to the best of our knowledge it is the first to leverage such representations for tree learning. The main contributions of this work are summarized as follows:

- This work introduces a neuro-symbolic framework de-

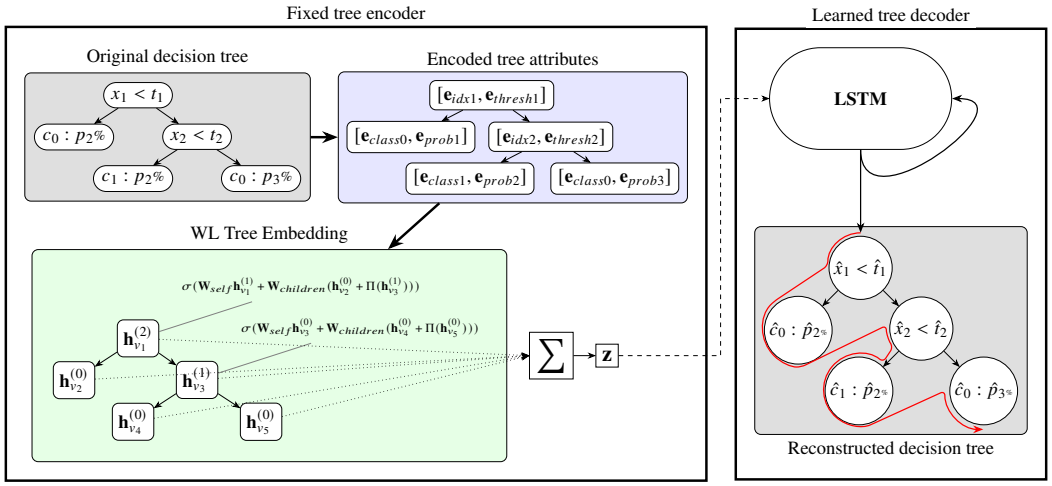


Fig. 1: Neural architecture for tree embedding and reconstruction. The node attributes of a decision tree are first embedded into fixed-size representations using random and periodic embeddings. Each node representation is then iteratively updated based on its children's representations through a series of aggregation and non-linear transformation steps; these representations are aggregated via sum-pooling, resulting in a fixed-size embedding for the entire tree. The embedding is then used to initialize an Long Short-Term Memory (LSTM) network that reconstructs the original tree structure by generating node attributes sequentially.

signed to enable federated learning of decision tree models.

- TreeClimber employs a neural architecture to obtain a general-purpose invertible mapping from discrete decision trees to a continuous latent space and back. This mechanism allows the model to bridge the gap between symbolic tree structures and the numerical optimization techniques used in federated learning.
- Given the non-differentiable nature of decision trees, TreeClimber utilizes a gradient-free optimization strategy to obtain a zeroth-order gradient approximation on each client, which can be effectively aggregated at the server level to update the global model.
- In contrast to existing federated tree learning methods, TreeClimber optimizes the structure and parameters of decision trees through a process analogous to FL, rather than heuristically assembling local models or parts thereof.
- An extensive experimental evaluation demonstrates that TreeClimber achieves competitive performance compared to traditional centralized training methods while ensuring data privacy.

The remainder of this work is organized as follows. Section II describes the neural architecture used to embed and reconstruct decision trees. Section III details the federated optimization strategy employed to collaboratively train the model. Section IV presents the experimental evaluation of TreeClimber, analyzing the trade-offs of the federated training process. Finally, Section V concludes the paper and outlines future research directions.

II. Neural Architecture

TreeClimber creates an invertible mapping between decision trees and a continuous latent space using a combination of randomized tree embeddings and neural networks. A high-level overview of this process is illustrated in Fig. 1, which shows the two main components: a fixed tree embedding module and a learned tree reconstruction module. The reconstruction module is trained to learn the grammar of decision trees, enabling it to reconstruct arbitrary tree structures from their embeddings without requiring dataset-specific fine-tuning.

A. Randomized tree embedding

TreeClimber employs a randomized version of the Weisfeiler-Lehman (WL) subtree graph embedding [14] to convert decision trees into fixed-size continuous vectors. The WL embedding process involves iteratively updating node representations based on their neighbors (child nodes in the case of trees) through hashing and aggregation operations, capturing the structural information of the tree over multiple iterations.

Each node in the decision tree is uniquely identified by a set of attributes $\mathcal{V} = \{idx, thresh, children\}$, where idx is the index of the feature used for splitting, $thresh$ is the threshold value for the split, and $children = \{left, right\}$ are the left and right child nodes. In leaf nodes, $left$ and $right$ are missing, while idx and $thresh$ have a different meaning, representing the class label and the probability of that class, respectively.

Initially, each node is assigned a representation that depends exclusively on its idx and $thresh$ attributes. The idx attribute embedding $\mathbf{e}^v_{idx} \in \mathbb{R}^d$ is obtained from a fixed random embedding matrix $\mathbf{E} \in \mathbb{R}^{(F+C)\times d}$, where F is the number of features, C is the number of classes, and d is the embedding dimension. These embeddings are sampled randomly from a Gaussian distribution and remain fixed. For the $thresh$ attribute, a periodic embedding is used to capture its continuous nature, following the approach proposed in [17] for numerical features.

Briefly, given a threshold value t, its periodic embedding $\mathbf{e}_t^v \in \mathbb{R}^{d'}$ is computed as per Eq. (1):

$$\big[\sin(2\pi t c_1), \cos(2\pi t c_1), \ldots, \sin(2\pi t c_{d'/2}), \cos(2\pi t c_{d'/2})\big], \quad (1)$$

where c_i are fixed frequencies sampled from a Gaussian distribution. The initial representation of a node v is then obtained by concatenating its idx and $thresh$ embeddings:

$$\mathbf{h}_v^{(0)} = \mathbf{e}_{idx} \parallel \mathbf{e}_{thresh}. \quad (2)$$

Subsequently, starting from the root node, the representations of each node are recursively updated by fusing the current node's representation and the representations of their child nodes through a non-linear transformation:

$$\mathbf{h}_v^{(t)} = \sigma\left(\mathbf{W}_{self}\mathbf{h}_v^{(t-1)} + \mathbf{W}_{children}\left(\mathbf{h}_{\text{left}(v)}^{(t-1)} + \Pi(\mathbf{h}_{\text{right}(v)}^{(t-1)})\right)\right), \quad (3)$$

where σ is a non-linear activation function (in this work tanh is used), $\Pi(\cdot)$ denotes a fixed random permutation (rotation) operator applied to the right child's representation before aggregation (which child to rotate is an arbitrary hyperparameter), and $\mathbf{W}_{self}$ and $\mathbf{W}_{children}$ are fixed random orthogonal matrices from $\mathbb{R}^{(d+d')\times(d+d')}$ such that their largest eigenvalue is lower than 1. This ensures that the local node representations are not dominated by either the node's own attributes or those of its children, and that the aggregation is sensitive to the order of left and right children. The choice of employing random fixed matrices and permutation is partly inspired by the hyperdimensional computing paradigm [18], and it has been found to be sufficient to capture the structural properties of the trees and graphs. In this work, one iteration of the algorithm was found to be sufficient. Afterwards, the final representation of the entire tree is obtained by aggregating the representations of all nodes in the tree through sum-pooling:

$$\mathbf{z} = \sum_{v \in \text{nodes}} \mathbf{h}_v^{(T)}. \quad (4)$$

The WL embedding process is summarized in Algorithm 1.

Algorithm 1 Decision tree embedding algorithm

Require: Decision tree $\mathcal{T}$, number of iterations T
Ensure: Tree embedding $\mathbf{z}$

for each node v in $\mathcal{T}$ **do**
  Compute $\mathbf{e}_{idx}^v$ from fixed random embedding matrix
  Compute $\mathbf{e}_{thresh}^v$ using periodic embedding
  Initialize $\mathbf{h}_v^{(0)} = \mathbf{e}_{idx} \parallel \mathbf{e}_{thresh}$
  Initialize matrices $\mathbf{W}_{self}$ and $\mathbf{W}_{children}$
for $t = 1$ to T **do**
  $frontier \leftarrow root\ of\ \mathcal{T}$
  while $frontier \neq \emptyset$ **do**
    $v \leftarrow frontier.pop()$
    $\mathbf{h}_v^{(t)} = \sigma\left(\mathbf{W}_{self}\mathbf{h}_v^{(t-1)} + \mathbf{W}_{children}(\mathbf{h}_{\text{left}(v)}^{(t-1)} + \Pi(\mathbf{h}_{\text{right}(v)}^{(t-1)}))\right)$
    for each child u of v **do**
      $frontier.push(u)$
Compute $\mathbf{z} = \sum_{v \in \text{nodes}} \mathbf{h}_v^{(T)}$
return $\mathbf{z}$

While the WL embedding is not directly invertible, it has been shown to be effective in capturing the structural properties of complex graphs [14]. Inspired by existing neuro-symbolic approaches for spatio-temporal logic [19], TreeClimber employs a neural network to learn the inverse mapping from the WL embedding space back to the decision tree structure. It is worth stressing that, in contrast with existing approaches, both the embedding and inverse mapping of TreeClimber do not depend on a specific dataset.

B. Neural tree reconstruction

TreeClimber employs a neural tree reconstruction module to learn the inverse mapping from the WL embedding space back to the symbolic decision tree structure. This module is implemented as a recurrent neural network (RNN), specifically a Long Short-Term Memory (LSTM) architecture, which reconstructs a sequence of tree nodes conditioned on an embedding vector $\mathbf{z}$.

The reconstruction process proceeds as follows. The embedding $\mathbf{z}$ is used to initialize the hidden state of the LSTM. At each step, the LSTM outputs a representation that is fed into two separate heads: a classification head that predicts the feature index idx for the current node, and a regression head that predicts the threshold value $thresh$. These predicted values are then re-embedded using the same embedding schemes as in the WL encoder (i.e., random embedding for idx and periodic embedding for $thresh$), and concatenated to form the input for the next LSTM step.

The tree nodes are generated following a pre-order traversal strategy, where the first generated node corresponds to the root of the tree; all subsequent nodes are attached as left children until a leaf node is generated (i.e., the predicted idx corresponds to an output class rather than a feature), at which point the next node is attached as the right child of the most recent non-leaf ancestor. The tree structure is thus implicitly defined by the sequence of generated nodes and their attributes, and does not require explicit representation of parent-child relationships. This process continues until all nodes either have two children or are designated as leaf nodes. The right part of Fig. 1 illustrates the order in which nodes are generated during the reconstruction process.

The reconstruction loss is defined as the sum of the cross-entropy loss for idx and the mean squared error for $thresh$, computed over all nodes in the tree. During training, teacher forcing is employed, where the ground-truth attributes are used as inputs to the LSTM at each step to stabilize learning.

In order to ensure that TreeClimber can generalize across different datasets without requiring dataset-specific fine-tuning, and that no hidden biases are introduced in the tree reconstruction process, the neural architecture is not trained on trees from specific datasets. Instead, training is performed on a large corpus of randomly generated decision trees. These trees are generated using a stochastic process that randomly selects feature and class indices and threshold values, ensuring a diverse set of tree structures, depths and configurations. The generation process is controlled by hyperparameters such as

maximum tree depth and number of features/classes, which are set to cover a wide range of possible scenarios. If a feature is chosen more than once along a path from the root to a leaf, the threshold values are adjusted to ensure that each split is valid and non-redundant. By training on a diverse set of randomly generated trees, the model learns to capture the underlying structural grammar of decision trees, without relying on possible correlations between features in real datasets, enabling it to generalize effectively to real-world datasets with no dataset-specific fine-tuning. This training process enables the neural architecture to learn a general-purpose mapping that can be effectively applied to any dataset during federated learning in a zero-shot manner, capturing tree topology regardless of the semantic meaning of the features.

III. Federated tree optimization

In federated learning with neural networks, clients typically compute gradients of the loss function with respect to model parameters using backpropagation. These gradients are then aggregated (usually through averaging) by the server to compute a global model update which is an estimate of the true gradient over the combined data of all clients. However, decision trees are inherently non-differentiable models, and the reconstruction process involves discrete decisions (e.g., selecting feature indices and threshold values), making it unfeasible to compute gradients directly through backpropagation. To overcome this challenge, TreeClimber employs an optimization strategy based on a zeroth-order approximation of the gradient [15].

In this approach, each client approximates the gradient of the loss function with respect to the tree embedding $\mathbf{z}$ by evaluating the loss at perturbed versions of $\mathbf{z}$. Specifically, for a given embedding $\mathbf{z}$, a set of m random perturbations $\{\mathbf{u}_i\}_{i=1}^m$ is sampled from a standard normal distribution. The client then evaluates the loss function $L(\mathbf{z} + \delta\mathbf{u}_i)$ for each perturbed embedding, where δ is a small scalar controlling the magnitude of the perturbation, and the loss is computed on the decision tree reconstructed from the perturbed embedding. The zeroth-order gradient approximation is computed as:

$$\hat{\nabla} L(\mathbf{z}) = \frac{1}{m} \sum_{i=1}^{m} \frac{L(\mathbf{z} + \delta\mathbf{u}_i) - L(\mathbf{z})}{\delta} \mathbf{u}_i. \tag{5}$$

All the clients compute this approximation based on their local data and send the estimated gradients to the server. Each perturbation requires reconstructing a decision tree from the perturbed embedding through a forward pass of the LSTM decoder, and evaluating its performance on the local dataset. The server aggregates the received gradients (e.g., by averaging) to compute a global gradient estimate, which is then used to update the global tree embedding $\mathbf{z}$ through a step of gradient descent. This process is repeated iteratively, allowing the clients to collaboratively optimize the decision tree model without requiring differentiability in the model structure or directly sharing their data. Nevertheless, model inversion attacks are a theoretical risk, as with all FL approaches; however, this risk is mitigated by the fact that only the tree embeddings and not the raw data or model parameters are shared, adding an additional layer of abstraction. The loss function used during optimization can be computed starting from any suitable metric for decision tree performance, such as classification accuracy or F1-score, depending on the specific application and dataset characteristics. In this work, the cross-entropy loss on the reconstructed tree's predictions is used for classification tasks, so that the decision tree can directly substitute a neural network classifier in typical federated learning pipelines.

Algorithm 2 Federated tree optimization

Require: Number of communication rounds R, number of clients K, number of perturbations m, perturbation magnitude δ, learning rate η, local datasets $\{\mathcal{D}_k\}_{k=1}^K$
Ensure: Global tree embedding $\mathbf{z}$

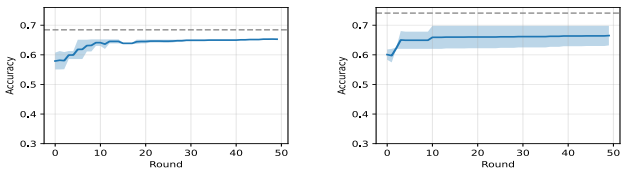
for each client $k = 1$ to K **in parallel do**
  $\mathcal{T}_k \leftarrow$ Train local decision tree on $\mathcal{D}_k$
  $\mathbf{z}_k \leftarrow$ WL embed $\mathcal{T}_k$, sent to server
Initialize global embedding $\mathbf{z} = \frac{1}{K} \sum_{k=1}^{K} \mathbf{z}_k$
for round $r = 1$ to R **do**
  for each client $k = 1$ to K **in parallel do**
    Receive global embedding $\mathbf{z}$
    Compute local loss $L_k(\mathbf{z})$ on local data
    for $i = 1$ to m **do**
      Sample random perturbation $\mathbf{u}_i \sim \mathcal{N}(0, I)$
      Compute perturbed loss $L_k(\mathbf{z} + \delta\mathbf{u}_i)$
    Compute local gradient approximation:
    $\hat{\nabla} L_k(\mathbf{z}) = \frac{1}{m} \sum_{i=1}^{m} \frac{L_k(\mathbf{z}+\delta\mathbf{u}_i) - L_k(\mathbf{z})}{\delta} \mathbf{u}_i$
    Send $\hat{\nabla} L_k(\mathbf{z})$ to server
  Aggregate and update: $\mathbf{z} \leftarrow \mathbf{z} - \eta \left(\frac{1}{K} \sum_{k=1}^{K} \hat{\nabla} L_k(\mathbf{z}) \right)$
return $\mathbf{z}$

The overall federated optimization process is summarized in Algorithm 2.

IV. Experimental evaluation

In order to assess the effectiveness of TreeClimber, an extensive experimental evaluation was conducted on two benchmark datasets for tabular machine learning tasks, specifically *Default of Credit Card Clients (card)* [20] and *Predict Students' Dropout and Academic Success (students)* [21], which are representative of real-world decision-support systems in finance and education, where interpretable models are often preferred and data privacy is a critical concern.

The aim of the experiments is to compare the performance of TreeClimber against traditional centralized decision tree training, not to obtain the best possible performance on each dataset; thus, to avoid spurious confounding factors, a single tree is used for classification instead of relying on ensemble methods such bagging or boosting. Given the stochastic nature of the federated optimization process, each experiment is repeated multiple times with different random seeds, and the average results are reported. This work proposes a preliminary feasibility study of the neurosimbolic federated tree learning paradigm, thus the experimental evaluation focuses on analyzing the behavior of TreeClimber, rather than performing an exhaustive comparison



(a) Dataset *card* (b) Dataset *students*

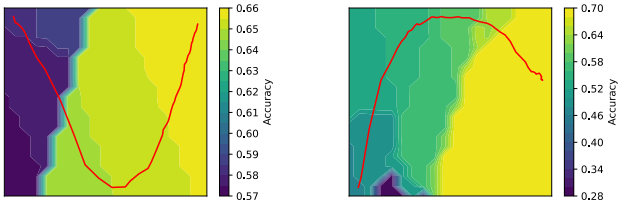
Fig. 2: Model performance over the number of communication rounds. The solid line represents the performance of TreeClimber, while the dashed line represents the performance of a centralized decision tree model trained on the same dataset.

with other federated tree learning methods or optimizing for the best possible performance on each dataset.

A. TreeClimber performance

The first set of experiments aims to assess the performance of TreeClimber compared to traditional centralized decision tree training. Figures 2a and 2b show how the performance of TreeClimber converges close to that of a centralized decision tree model as the number of communication rounds increases. The results demonstrate that TreeClimber is able to effectively learn decision trees in a federated setting, achieving competitive performance compared to centralized training methods. The convergence behavior is consistent across the two datasets, indicating that TreeClimber can effectively optimize the tree embeddings through the federated optimization process.

Fig. 3 shows the variation of the accuracy landscape across the two principal components of the tree embedding space during the optimization process by evaluating the accuracy of the trees whose embeddings lie on a grid of points along the principal components, and the results are visualized using a heatmap. The figure shows a reasonably well-behaved space, and TreeClimber successfully navigates it, moving from a lower-accuracy area (showing limited mode connectivity with the initial tree averaging) to a flat, higher-accuracy one.

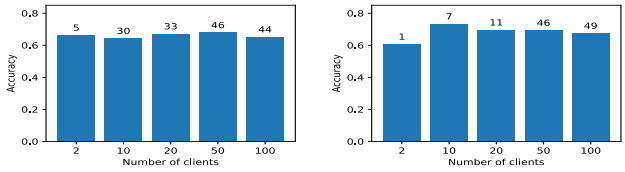


(a) Dataset *card* (b) Dataset *students*

Fig. 3: Accuracy landscape visualization across the two principal components of the tree embedding space during the optimization process. The red line represents the trajectory followed by the optimization process.

B. Scalability to multiple clients

The next set of experiments aims to assess the scalability of TreeClimber to multiple clients. In particular, the setting is varied from a data silos scenario, where the dataset is partitioned into two clients, to more challenging scenarios with up to 100 clients. Fig. 4 shows the final model performance as the number of clients increases. The results indicate roughly consistent performance across the number of clients, with slight benefits for the binary classification tasks of the *card* dataset as it increases the number of samples used to estimate the gradient, and maintains competitive performance even in highly distributed scenarios. It is worth noting that as the number of clients increases, the number of communication rounds required to reach convergence increases, as smaller local datasets in highly distributed scenarios make for noisier updates. Similar behavior is observed in traditional federated learning with neural networks.



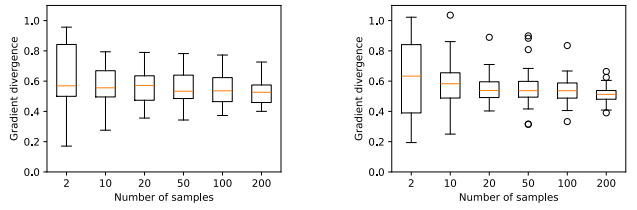
(a) Dataset *card* (b) Dataset *students*

Fig. 4: Final model performance varying the number of clients in the federated setting. Above each bar, the number of communication rounds required for convergence is reported.

C. Gradient approximation sensitivity

The zeroth-order gradient approximation is a crucial component of TreeClimber, as it allows the model to optimize the tree embedding without requiring direct access to the data. The choice of the number of perturbations m and the perturbation magnitude δ can significantly impact the quality of the gradient approximation. As show in Fig. 5, the number of perturbations in particular determines the trade-off between computational cost and accuracy of the gradient estimate. To assess the quality of the gradient estimate, 100 different approximations were computed for each number of perturbations m, and the cosine distance of the estimated gradients and the overall average was computed. Higher values of m lead to more accurate gradient estimates, with smaller variability in the gradient direction w.r.t. the true gradient, leading to a smoother convergence behavior. On the other hand, the computational cost increases linearly with m, as each perturbation requires reconstructing a decision tree and evaluating its performance on the local dataset.

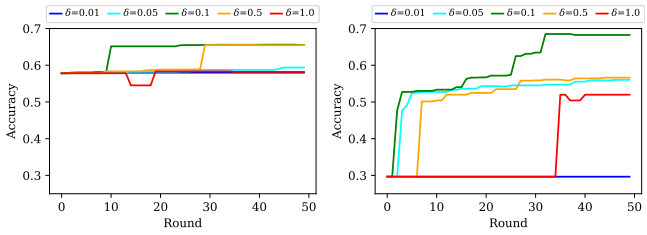
Additionally, the perturbation magnitude δ plays a critical role in determining the accuracy of the gradient approximation. If δ is too small, the perturbations may not sufficiently explore the embedding space, leading to noisy gradient estimates. Conversely, if δ is too large, the perturbations may deviate too far from the current embedding, resulting in inaccurate gradient estimates. Fig. 6 illustrates the impact of varying δ on the training speed and converged performance of TreeClimber. The number of perturbations is fixed to $m = 100$ in these experiments, as this value provided a good balance between



(a) Dataset *card* (b) Dataset *students*

Fig. 5: Training behavior varying the number of perturbations m used for the gradient approximation. The number of perturbations impacts the stability of the optimization process. Fewer perturbations introduce more noise during training, making the optimization less stable.

computational cost and gradient accuracy in preliminary tests. The results indicate that there exists an optimal range for δ that balances exploration and accuracy in the gradient approximation, leading to faster convergence and better model performance. In this case, the optimal δ appears to be 0.1, as values below this threshold lead to slower convergence due to insufficient exploration, while values above this threshold result in erratic optimization behavior.



(a) Dataset *card* (b) Dataset *students*

Fig. 6: Test accuracy over rounds varying the perturbation magnitude δ. Excessively small perturbations cannot effectively update the model, while a too large δ leads to unstable training.

V. Conclusion

While Federated Learning has been widely adopted in recent years, most of the proposed solutions are based on deep learning models, which often lack interpretability and may not be suitable for all types of data. This paper presented TreeClimber: a novel neuro-symbolic approach that leverages gradient-free optimization techniques to train decision tree models in a federated setting. The proposed method allows for the training of interpretable models, facilitating human-in-the-loop deployment, while preserving data privacy across distributed clients, making it a valuable tool for automated decision-making in distributed systems. Experimental results demonstrate the effectiveness of the approach, obtaining competitive performance compared to traditional centralized training methods. This opens new avenues for deploying interpretable machine learning models in privacy-sensitive applications. Future work will focus on extending the neural architecture to support better shaping of the embedded space, and developing additional optimization algorithms to enhance the training process in federated environments.

References

[1] 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.

[2] A. Augello, G. Falzone, and G. Lo Re, "Dcfl: Dynamic clustered federated learning under differential privacy settings," in *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, 2023, pp. 614–619.

[3] F. Concone, C. Ferdico, G. Lo Re, and M. Morana, "A federated learning approach for distributed human activity recognition," in *2022 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2022, pp. 269–274.

[4] N. Seidi, S. Roy, S. K. Das, and A. Tripathy, "Addressing data heterogeneity in federated learning of cox proportional hazards models," in *2024 IEEE International Conference on E-health Networking, Application & Services (HealthCom)*. IEEE, 2024, pp. 1–7.

[5] A. Augello, A. De Paola, and G. Lo Re, "M2fd: Mobile malware federated detection under concept drift," *Computers & Security*, p. 104361, 2025.

[6] S. Bharati, M. R. H. Mondal, and P. Podder, "A review on explainable artificial intelligence for healthcare: Why, how, and when?" *IEEE Trans. on Artificial Intelligence*, vol. 5, no. 4, pp. 1429–1442, 2023.

[7] A. De Paola, P. Ferraro, G. Lo Re, M. Morana, and M. Ortolani, "A fog-based hybrid intelligent system for energy saving in smart buildings," *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, no. 7, pp. 2793 – 2807, 2020.

[8] F. Concone, G. Lo Re, and M. Morana, "A fog-based application for human activity recognition using personal smart devices," *ACM Transactions on Internet Technology*, vol. 19, no. 2, 2019.

[9] A. Augello, A. Gupta, G. Lo Re, and S. K. Das, "FairRFL: Fair and robust federated learning in the presence of selfish clients," *IEEE Transactions on Emerging Topics in Computing*, vol. 14, no. 1, pp. 316–331, 2026.

[10] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," *Information Fusion*, vol. 81, pp. 84–90, 2022.

[11] Z. Tian, R. Zhang, X. Hou, L. Lyu, T. Zhang, J. Liu, and K. Ren, "FederBoost: Private federated learning for GBDT," *IEEE Trans. on Dependable and Secure Computing*, vol. 21, no. 3, pp. 1274–1285, 2023.

[12] A. Khraisat, M. A. Talukder, M. A. Uddin, and A. Alazab, "Rf-fedavg: Federated learning-based random forest model for intrusion detection in wireless sensor networks," *Cluster Computing*, vol. 28, no. 13, p. 873, 2025.

[13] P. A. E. Lim and C. H. Park, "A collaborative ensemble construction method for federated random forest," *Expert Systems with Applications*, vol. 255, p. 124742, 2024.

[14] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt, "Wasserstein weisfeiler-lehman graph kernels," *Advances in neural information processing systems*, vol. 32, 2019.

[15] K. Balasubramanian and S. Ghadimi, "Zeroth-order nonconvex stochastic optimization: Handling constraints, high dimensionality, and saddle points," *Foundations of Computational Mathematics*, vol. 22, no. 1, pp. 35–76, 2022.

[16] F. Spinnato, A. Matropietro, and R. Guidotti, "Implicit neural decision trees," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning.*, 2025, pp. 585–590.

[17] Y. Gorishniy, I. Rubachev, and A. Babenko, "On embeddings for numerical features in tabular deep learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 991–25 004, 2022.

[18] A. Augello, A. De Paola, G. Lo Re, and A. Menager, "HDDroid: Federated hyperdimensional computing for mobile malware detection," in *CEUR Workshop Proceedings*, 2025.

[19] S. Candussio, G. Saveri, G. Sarti, and L. Bortolussi, "Bridging logic and learning: Decoding temporal logic embeddings via transformers," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2025, pp. 3–18.

[20] I.-C. Yeh, "Default of Credit Card Clients," UCI Machine Learning Repository, 2009, DOI: https://doi.org/10.24432/C55S3H.

[21] V. Realinho, M. Vieira Martins, J. Machado, and L. Baptista, "Predict Students' Dropout and Academic Success," UCI Machine Learning Repository, 2021, DOI: https://doi.org/10.24432/C5MC89.