MOTHERTREE: META-LEARNING ON SYNTHETIC DATA IMPROVES DECISION TREE TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Conventional decision tree algorithms produce effective, transparent models that can be audited, communicated, and deployed independently of the training data, but require learning every new task from scratch. In contrast, tabular foundation models demonstrate that meta-learning from a synthetic prior distribution enables strong in-context prediction for previously unseen tasks, especially in small-sample regimes. However, this approach does not produce a standalone model that can be inspected in isolation. We introduce MotherTree, a tabular transformer that meta-learns decision tree induction: given a training set for a new task, it outputs a hard, axis-aligned decision tree, equivalent in form to classically trained trees, in a single forward pass. MotherTree is pre-trained on a synthetic prior using stochastic gradient descent without requiring reference trees for supervision. On established benchmarks with controlled sample size, the approach is competitive with size-matched trees from common algorithms: recursive partitioning, gradient-based tree learning, globally optimal trees, and distillation from tabular foundation models. Notably, MotherTree consistently improves over from-scratch gradient-based learning and acts as a strong initializer: task-specific tuning of the generated tree outperforms the corresponding from-scratch learner on all benchmarks and sample sizes. These results show that meta-learning can provide effective inductive biases for learning stand-alone, small decision tree classifiers.

1 INTRODUCTION

Small decision trees are critical components of prediction and decision systems (Rudin et al., 2022). They are one of few model classes that can be directly inspected, audited, and communicated (Rudin, 2019). Still, tree models are usually trained one by one without transfer between learning tasks, failing to benefit from similar tasks being solved before. Tabular foundation models (TFM) (Hollmann et al., 2023) suggest a different possibility by demonstrating the utility of meta-learning for tabular prediction. By pre-training on numerous data sets from a synthetic prior, TFMs have proven remarkably efficient at learning new tasks from small samples (Hollmann et al., 2025). Inspired, we ask: How can decision tree induction benefit from synthetic-prior meta-learning?

Current tabular foundation models like TabICL (QU et al., 2026) and TabPFN (Hollmann et al., 2025) exploit in-context learning (Brown et al., 2020), using Transformers (Vaswani et al., 2017) to map directly from a labeled training set to test set predictions. After pre-training on tasks from a synthetic prior, in-context TFMs perform the equivalent of both training and prediction for unseen tasks in a single forward pass. This strategy has shown tremendous potential in benchmarks (Erickson et al., 2026), but generalization challenges remain and classical learners, including tree-based models, are still competitive in small samples and preferable in large-sample regimes (Holzmüller et al., 2024; Purucker et al., 2026). Moreover, in-context learning does not produce a reusable stand-alone model for a specific task, creating obstacles for interpretation and model sharing.

Decision tree algorithms that use recursive partitioning (Breiman et al., 1984; Quinlan, 1993), greedily applying hand-crafted splitting rules, remain the default choice for many (scikit-learn, 2026), but do not produce optimal trees and often overfit in small samples. Optimal tree algorithms (Bertsimas & Dunn, 2017; Aglin et al., 2020) perform especially well in small-model scenarios but are highly computationally intensive and difficult to scale. Gradient-based tree learning (Yang et al., 2018; Marton et al., 2024) fits all splits and leaf values jointly but has no guarantee of optimality, and has

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

yet to demonstrate generalizable superiority over greedy methods. All three classes of algorithms could benefit from improving either their sample efficiency or the optimization itself. First attempts have been made to *meta-learn* tree fitting algorithms (Zhuang et al., 2024), but rely on access to gold-standard reference trees for supervision. Scalable and efficient meta-learning of decision trees remains a largely open problem. We make the following contributions toward solving it:

- We formulate decision tree induction as an amortized meta-learning problem and introduce MotherTree, a transformer model that generates standalone hard axis-aligned decision tree classifiers in a single forward pass. MotherTree uses a tree-structured decoder, trainable without reference trees, enabling large-scale pre-training on data sets from a diverse synthetic prior distribution, and generalization to real-world tasks with different feature sets.
- We evaluate MotherTree on 64 benchmark classification tasks from TabArena and OpenML CC-18, in controlled small-sample regimes, comparing against greedy, optimal, gradient-based, and distillation-based tree learners. MotherTree consistently outperforms the corresponding from-scratch learner, GradTree (Marton et al., 2024), e.g., exceeding its AUC by 0.069 on TabArena and 0.077 on OpenML CC-18 for $B = 100$ training samples (Figure 2).
- By fine-tuning the generated trees on task-specific data (MotherTree-Tune), we create additional performance gains that grow with the size of the training set, making the models competitive with all four baseline classes of tree learners. Synthetic experiments on learning noiseless parity functions suggest that the utility of MotherTree lies not only in improved sample-efficiency but in improved initialization for local optimization.

2 Related work

Our goal is to amortize learning of hard, axis-aligned decision trees. *Hard* trees use discrete, non-differentiable rules and deterministic routing, and *soft* trees use continuous, differentiable rules with fuzzy or probabilistic routing; see surveys by Li et al. (2022); Costa & Pedreira (2023). *Oblique* trees (Costa & Pedreira, 2023; Mao et al., 2025; Norouzi et al., 2015) use linear combinations of feature values rather than axis-aligned splits formed by a single feature and threshold. Common learning strategies for hard trees include greedy recursive partitioning (Breiman et al., 1984; Quinlan, 1993) and global discrete optimization (Bertsimas & Dunn, 2017; Hu et al., 2019; Aglin et al., 2020). Both hard and soft trees may be parameterized as neural networks and trained using stochastic gradient descent (Tanno et al., 2019; Yang et al., 2018; Li et al., 2022; Mao et al., 2025). Soft trees use fully differentiable parameterizations (Pace et al., 2022; Irsoy et al., 2012; Tanno et al., 2019) but require post-hoc rounding to form hard, axis-aligned trees. Moreover, gradient-descent strategies are sensitive to initialization for both hard and soft trees. What's more, all three classes of algorithms above fit trees from scratch, failing to benefit from solving such tasks before.

The goal of meta-learning is to fit a base model that can quickly learn a new task from a small sample of task-specific data (Schmidhuber, 1992; Ravi & Larochelle, 2017; Finn et al., 2017). MotherNet (Müller et al., 2025) applied hypernetworks (Ha et al., 2016) to meta-learning for tabular prediction, training a transformer to take a small tabular data set as input and output the parameters of a task-specific multi-layer perceptron (MLP). Mahfoud et al. (2025) considered tree learning as "amortized structure inference", but not in the sense of meta-learning from pretraining tasks. Knauer et al. (2025) use a large language model to output a decision tree based purely on metadata, not on tabular samples. Distillation (Frosst & Hinton, 2017) from a tabular foundation model can also meta-learn trees. Our closest related work is MetaTree (Zhuang et al., 2024), a Transformer for generating tree classifiers, pre-trained with supervision from ground-truth reference trees. MetaTree is limited by the need to construct high-quality ground-truth trees for each pre-training task. In contrast, MotherTree learns directly from instance labels, which proved preferable in our experiments.

3 MotherTree: Meta-learned decision tree training

We propose MotherTree—a meta-learned decision tree generator. The model combines a tabular data encoder with a tree-structured decoder to learn a map M_ϕ from labeled training set $\mathcal{D}$ to the parameters of an axis-aligned decision tree classifier, pre-trained end-to-end with instance labels as supervision. See Figure 1. For a new task, MotherTree uses a single forward pass to fit a tree that can be further tuned if desired. In this way, the map M_ϕ acts exactly like a tree learning algorithm.

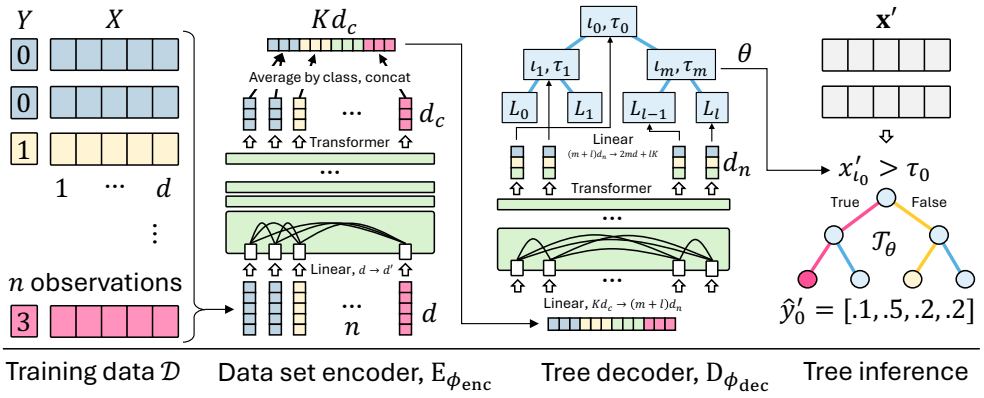


Figure 1: When applying MotherTree to a new task, labeled training data is fed to the data set encoder to generate sample embeddings that are aggregated by class, forming a single data set embedding. The tree decoder maps the data set embedding to one embedding per node in the generated tree using a transformer, ultimately predicting the node split (ι_i, τ_i) (for internal nodes) or value L_j (for leaves). These define an axis-aligned, deterministic tree for stand-alone inference.

3.1 Meta-learning decision tree parameters

Let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ denote a labeled training set for a classification problem (task) $\mathcal{P}$, where $\mathbf{x}_i \in \mathbb{R}^d$ denotes the input features and $y_i \in \{1, \ldots, K\}$ the labels of an instance i. Our goal is to *learn* a procedure, by solving pre-training problems $\mathcal{P}_1, ..., \mathcal{P}_N$, that can take the unseen data set $\mathcal{D}$ and construct a hard, axis-aligned decision tree classifier, with small error on new examples from $\mathcal{P}$. We approach this problem by constructing decision trees based on a continuous parameterization.

Consider a complete binary tree with internal nodes $\mathcal{I}$ and leaf nodes $\mathcal{L}$, where $m = |\mathcal{I}|$ and $l = |\mathcal{L}|$. Following GradTree (Marton et al., 2024), we characterize the tree using parameters $\theta = (I, T, L)$, where $I, T \in \mathbb{R}^{m \times d}$ represent the feature selection indices and thresholds of internal nodes, and $L \in \mathbb{R}^{l \times K}$ the leaf values. The topology of the tree is kept fixed as a complete binary tree with a given depth D. The splitting rule s_j of each internal node $j \in \mathcal{I}$ is determined by

$$\iota_j = \arg\max_{r \in \{1,\ldots,d\}} I_{j,r}, \qquad \tau_j = T_{j,\iota_j}, \qquad s_j(\mathbf{x}) = \mathbb{1}[x_{\iota_j} \leq \tau_j], \tag{1}$$

such that $s_j(\mathbf{x})$ routes the sample $\mathbf{x}$ to j:s first child if true, and right otherwise. Just like standard decision trees, a tree $\mathcal{T}_\theta$ parameterized by $\theta = (I, T, L)$, routes each input $\mathbf{x}$ deterministically to a leaf $l(\mathbf{x})$ and outputs predicted class probabilities $\mathcal{T}_\theta(\mathbf{x}) = L_{l(\mathbf{x})} \in \Delta^{K-1}$.

Learning a tree for a problem $\mathcal{P}$ amounts to choosing parameters θ such that the classification error of the induced tree $\mathcal{T}_\theta$ is minimized on $\mathcal{P}$. MotherTree amortizes learning across tasks by fitting a reusable *tree generator* $\mathrm{M}_\phi : \mathcal{D} \mapsto \theta$, whose parameters ϕ are fit on tasks from a synthetic prior. Given a training set $\mathcal{D}_{\text{new}}$ for a new task $\mathcal{P}_{\text{new}}$, parameters θ_{new} and predictions $\hat{y}$ are computed by

$$\theta_{\text{new}} = (I_{\text{new}}, T_{\text{new}}, L_{\text{new}}) = \mathrm{M}_\phi(\mathcal{D}_{\text{new}}), \qquad \hat{y} = \mathcal{T}_{\theta_{\text{new}}}(\mathbf{x}), \text{for an instance } \mathbf{x} \text{ of } \mathcal{P}_{\text{new}}.$$

Once parameters θ are generated, the tree $\mathcal{T}_{\theta_{\text{new}}}$ can be used without access to M_ϕ or $\mathcal{D}_{\text{new}}$.

3.2 Tabular data set encoder

The MotherTree generator M_ϕ with parameters $\phi = (\phi_{\text{enc}}, \phi_{\text{dec}})$ composes a tabular data set encoder $\mathrm{E}_{\phi_{\text{enc}}}$ and a tree-structured decoder $\mathrm{D}_{\phi_{\text{dec}}}$. Given a labeled training set $\mathcal{D}$ of n rows, the encoder first embeds $\mathcal{D}$ into row-wise representations $H = [\mathbf{h}_1, \ldots, \mathbf{h}_n]^\top \in \mathbb{R}^{n \times d_c}$, where d_c is a hyperparameter. In the default setting, H is computed using the MotherNet (Müller et al., 2025) encoder, which applies a multi-layer transformer with row-wise attention to a linear projection of the input features (see Appendix A.1 for further details). The row representations are then aggregated by class and concatenated into a single vector-valued data set embedding,

$$\mathrm{E}_{\phi_{\text{enc}}}(\mathcal{D}) = [\overline{\mathbf{h}}_1; \ldots; \overline{\mathbf{h}}_K] \in \mathbb{R}^{K d_c} \quad \text{where} \quad \forall c \in [K] : \overline{\mathbf{h}}_c = \frac{1}{|\mathcal{I}_c|} \sum_{i \in \mathcal{I}_c} \mathbf{h}_i, \quad \mathcal{I}_c = \{i : y_i = c\}.$$

This embedding completes the encoder $\mathrm{E}_{\phi_{\mathrm{enc}}}$ and serves as the input to the tree-structured decoder.

Encoding for standalone inference. Like tabular foundation models, e.g., Hollmann et al. (2025); QU et al. (2026), based on prior-fitted networks (PFN) (Müller et al., 2022), MotherTree is pre-trained on tasks from a synthetic prior, but differs fundamentally in how the model is used for inference. In-context PFN models jointly encode both training and test sets when making predictions, allowing test set embeddings to attend to the training set. In contrast, the MotherTree generator M_ϕ processes only the training set, acting as a hypernetwork that directly instantiates a task-specific decision tree. Consequently, *there is no test-to-training-set attention*. Once generated, the tree is a stand-alone model; test-time prediction requires only tree traversal, but no access to the training set or the tree generator. This distinction changes the role of the data set encoder. MotherTree requires the encoder to summarize the training set for model generation rather than to measure similarities between training and test samples. Consistent with this, replacing the MotherNet encoder with the more expensive TabICLv2 encoder does not improve performance; see Appendix A.2, suggesting that a stronger in-context prediction backbone does not translate into a better tree generator.

3.3 Ancestral decoder for axis-aligned trees

We introduce a *tree-aware transformer decoder* $\mathrm{D}_{\phi_{\mathrm{dec}}}$ to map the data set embedding to decision tree parameters, $\theta = \mathrm{D}_{\phi_{\mathrm{dec}}}(\mathrm{E}_{\phi_{\mathrm{enc}}}(\mathcal{D}))$. For a complete binary tree with m internal nodes and l leaves, we construct $N = m+l$ token pre-embeddings, $\mathbf{d}_1^{(0)}, ..., \mathbf{d}_N^{(0)}$, each corresponding to a specific node in the tree, by a linear projection of the data set embedding,

$$H^{(0)} \coloneqq [\mathbf{d}_1^{(0)}, \ldots, \mathbf{d}_N^{(0)}] = W_{\mathrm{emb}}\, \mathrm{E}_{\phi_{\mathrm{enc}}}(\mathcal{D}) + \mathbf{b}_{\mathrm{emb}} \in \mathbb{R}^{N \times d_n}, \qquad \forall i : \mathbf{d}_i^{(0)} \in \mathbb{R}^{d_n}.$$

The node pre-embeddings are passed through a multi-layer transformer before being used to output splitting parameters (I_j, T_j) for internal nodes j and leaf values L_k for leaves k. To encode the root-to-node dependency structure, we apply a masked attention with an *ancestral attention mask* M that allows each node to attend only to itself and its ancestors. Specifically,

$$\text{for nodes } i, j \in [N]: \quad M_{i,j} = \begin{cases} 0, & \text{if } \mathrm{Ancestor}(j, i), \\ -\infty, & \text{otherwise}, \end{cases} \tag{2}$$

where $\mathrm{Ancestor}(j, i)$ indicates that node j is an ancestor of node i, including i itself. This implicitly conditions each node on the splits of its ancestors while preventing information flow from unrelated branches of the tree. An ℓ-layer transformer with mask M as in equation 2 is applied to $H^{(0)}$, producing node embeddings $H^{(\ell)} = [\mathbf{d}_1^{(\ell)}, \ldots, \mathbf{d}_N^{(\ell)}]$. The decoder $\mathrm{D}_{\phi_{\mathrm{dec}}}$ is finalized by applying a linear layer to each embedding $\mathbf{d}_j^{(\ell)}$, producing $\theta_j = (I_j, T_j, L_j)$, and concatenating the results.

Comparison with MLP decoders. MotherNet (Müller et al., 2025) uses a simple MLP decoder $\theta = \mathrm{MLP}(\mathrm{E}_{\phi_{\mathrm{enc}}}(\mathcal{D}))$. While effective for generating the parameters of an MLP, this decoder does not explicitly model structural dependencies among tree parameters. In ablation experiments (Section 4.1), we find that ancestral decoding performs substantially better in MotherTree. Moreover, we do not utilize the low-rank strategy of the MotherNet decoder, as early ablations found no utility in this additional structure, likely partly due to the tree requiring fewer parameters than the MLP.

3.4 Pre-training, tuning, and inference

The MotherTree generator $\mathrm{M}_\phi = \mathrm{D}_{\phi_{\mathrm{dec}}} \circ \mathrm{E}_{\phi_{\mathrm{enc}}}$ is pre-trained (prior-fitted) on data sets from a large number of synthetic tasks, sampled from a structural causal model (SCM) prior $p_{\mathrm{syn}}(\mathcal{D})$. For each sampled pre-training data set $\mathcal{D}$, the generator is trained end-to-end through the tree by minimizing

$$\mathcal{L}_{\mathrm{meta}}(\phi; \mathcal{D}) = -\frac{1}{n} \sum_{i=1}^{n} \log p_{\mathrm{M}_\phi(\mathcal{D})}(y_i \mid \mathbf{x}_i), \qquad \phi^* = \arg\min_{\phi} \mathbb{E}_{\mathcal{D} \sim p_{\mathrm{syn}}} \left[\mathcal{L}_{\mathrm{meta}}(\phi; \mathcal{D}) \right].$$

The estimated probability of each class is given by the corresponding element of the tree output, $\hat{p}_{\mathrm{M}_\phi(\mathcal{D})}(y_i \mid \mathbf{x}_i) = \mathcal{T}_{\mathrm{M}_\phi(\mathcal{D})}(\mathbf{x}_i)_{y_i}$. We jointly optimize the parameters of the data set encoder and tree decoder, $\phi = (\phi_{\mathrm{dec}}, \phi_{\mathrm{enc}})$, across tasks using stochastic gradient descent. Unlike MetaTree (Zhuang et al., 2024), this does not rely on supervision from learned reference trees. Computing

$\mathcal{T}(\mathbf{x}_i)$, and therefore the learning objective, involves non-differentiable operations for both feature selection (arg max) and routing (threshold) as per equation 1. Following Marton et al. (2024), we overcome this by passing gradients using the straight-through operator for both operations.

Post-hoc parameter tuning. By default, we re-calibrate the generated leaf parameters L_{new} using task-specific $\mathcal{D}_{\text{new}}$, akin to Mao et al. (2025), by setting their values equal to the observed class proportions of samples reaching each leaf. Additionally, we propose a variant dubbed MotherTree-Tune that fine-tunes *all* generated parameters $\theta_{\text{new}} = (I_{\text{new}}, T_{\text{new}}, L_{\text{new}})$ to $\mathcal{D}_{\text{new}}$ using stochastic gradient descent, treating θ_{new} as free parameters. In this way, we can use the generated tree as an *initialization* for task-specific optimization using the same procedure as GradTree.

4 Experiments

We evaluate MotherTree in two tabular benchmarks and a synthetic task designed to illustrate the strengths and weaknesses of tree learning algorithms. We pay close attention to the comparison with GradTree as it uses the same tree parameterization but does not exploit meta-learning. Section 4.1 showcases MotherTree's meta-learning capabilities in the low-sample regime, especially in binary classification, and demonstrates that MotherTree provides a strong initialization for task-specific tuning (MotherTree-Tune). Ablations demonstrate the utility of our ancestral decoder. In Section 4.2, we study the strength of MotherTree as a tree optimizer in a synthetic task with known solution.

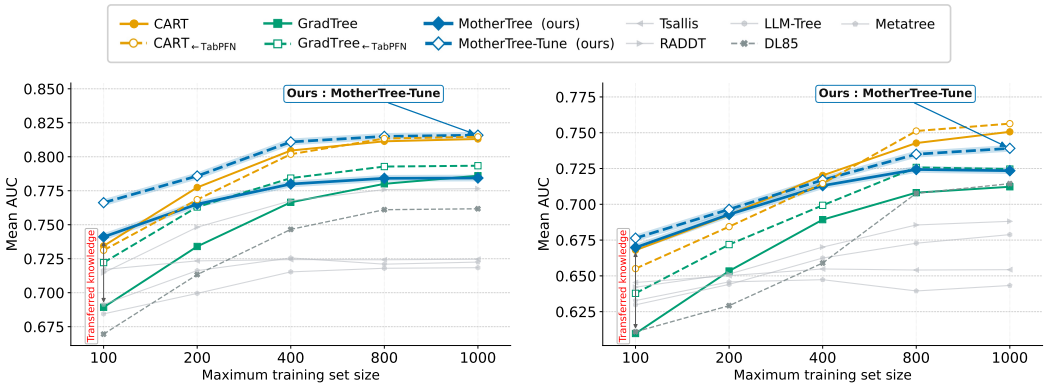
Baselines. We compare MotherTree to a suite of single-tree learning algorithms, including (i) Greedy and globally optimal trees: CART decision tree implemented in scikit-learn (Pedregosa et al., 2011), Tsallis (Balcan & Sharma, 2024) and DL85 (Aglin et al., 2020); (ii) Gradient-based trees: RADDT (Mao et al., 2025), GradTree (Marton et al., 2024), DNDT (Yang et al., 2018); (iii) Meta tree learner: MetaTree (Zhuang et al., 2024), LLM-Tree (Knauer et al., 2025); (iv) Models distilled from TabPFN2.6 (Hollmann et al., 2025): CART$_{\leftarrow \text{TabPFN}}$ and GradTree$_{\leftarrow \text{TabPFN}}$. Details regarding the distillation methods are stated in Appendix A.4.2. We use existing PyTorch implementations where available and re-implement other gradient-based baselines in PyTorch. Hyperparameters are tuned using grid search. Details on baselines can be found in Appendix A.4.3. Since we aim to compare the learning properties of different algorithms, rather than hyperparameter tuning, we fix the topology of all trees to have the same maximum depth, $D = 5$.

Implementation Details. MotherTree is pre-trained in two stages. In the first stage, we utilize the synthetic prior from TabICL (QU et al., 2025) with $400k$ training steps. For the second stage, the TabICLv2 (QU et al., 2026) prior is used for training $125k$ steps. The training batch size is set to 128 and 64, respectively. The maximum learning rate is set to be 3×10^{-5} with LinearLR scheduler and cosine annealing scheduler for each stage. Training was conducted on 4 $\times$ NVIDIA GH200 120GB. See Appendix A.3 for details and ablations for pre-training and hyperparameters.

Evaluation protocol. We evaluate all methods under a varying training set budget $B \in \{100, 200, 400, 800, 1000\}$, where B defines the *maximum* number of training examples available to the learner in a given task. For each of five random seeds, we shuffle the samples and construct a train-test split subject to this budget. Details for sampling, preprocessing, and missing value handling are shown in Appendix A.4.4 & A.4.5. When the data set size is insufficient to allocate B training examples while retaining a test set, we use a balanced train–test split instead. Performance (uncertainty) is measured using the ROC-AUC with average (standard dev.) computed over the five splits and the data sets within each benchmark. We report the average AUC ranks in critical difference diagrams, for which staistical significance is assessed using the Friedman test followed by the Nemenyi test, following the protocol in TabArena (Erickson et al., 2026). To evaluate the tree structure of gradient-based tree learners, we measure the number of uninformative paths—root-to-leaf paths on which any two splitting rules contradict each other. Finally, we report the leaf utilization—the fraction of leaf nodes which contain at least one sample, in a fixed draw of 500 test set samples.

4.1 Evaluation on tabular benchmarks

We evaluate all methods on 64 unique tabular classification data sets from two standard benchmarks: (i) 30 data sets from the small version of OpenML-CC18 (Bischl et al., 2017) (17 binary,



(a) OpenML-CC18-Small

(b) TabArena

Figure 2: Mean AUC for varying training set sample budget, B. See also Appendix Table 5.

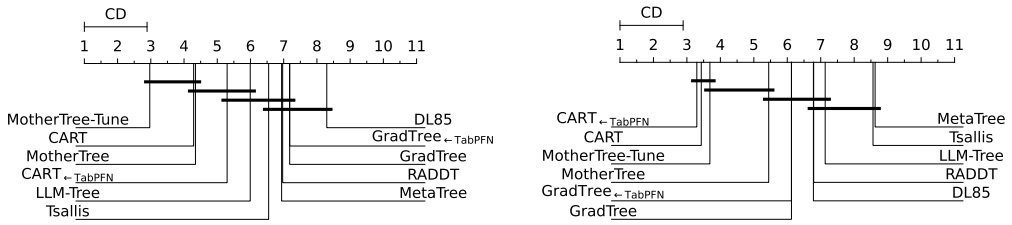
Table 1: Experimental results (avg. AUC ± std.) across tasks with training set budget $B = 200$ on two benchmarks. Tables for all sample size budgets can be viewed in Appendix A.5. GradTree and MotherTree share tree parameterization. Best results shown in **bold**, second-best results underlined.

		TabArena ($B = 200$)		OpenML-CC18-Small ($B = 200$)	
	Method	Binary	Multiclass	Binary	Multiclass
	CART	$0.670_{\pm 0.107}$	$\underline{0.779}_{\pm 0.170}$	$0.759_{\pm 0.123}$	$\mathbf{0.802}_{\pm 0.110}$
	CART$_{\leftarrow \text{TabPFN}}$	$0.659_{\pm 0.099}$	$\mathbf{0.780}_{\pm 0.165}$	$0.748_{\pm 0.125}$	$0.795_{\pm 0.105}$
	(α^*, β^*) Tsallis	$0.620_{\pm 0.108}$	$0.762_{\pm 0.167}$	$0.680_{\pm 0.155}$	$0.780_{\pm 0.115}$
	DNDT	$0.595_{\pm 0.079}$	$0.595_{\pm 0.055}$	$0.650_{\pm 0.108}$	$0.643_{\pm 0.076}$
	RADDT	$0.627_{\pm 0.100}$	$0.744_{\pm 0.152}$	$0.710_{\pm 0.135}$	$0.798_{\pm 0.124}$
	DL85	$0.604_{\pm 0.088}$	$0.722_{\pm 0.142}$	$0.691_{\pm 0.131}$	$0.743_{\pm 0.101}$
	MetaTree	$0.632_{\pm 0.085}$	$0.696_{\pm 0.145}$	$0.705_{\pm 0.141}$	$0.731_{\pm 0.103}$
	LLM-Tree	$0.641_{\pm 0.092}$	$0.654_{\pm 0.105}$	$0.669_{\pm 0.145}$	$0.739_{\pm 0.118}$
Same tree param. space	GradTree	$0.635_{\pm 0.081}$	$0.723_{\pm 0.141}$	$0.725_{\pm 0.132}$	$0.746_{\pm 0.118}$
Same tree param. space	GradTree$_{\leftarrow \text{TabPFN}}$	$0.655_{\pm 0.095}$	$0.715_{\pm 0.146}$	$0.752_{\pm 0.128}$	$0.774_{\pm 0.104}$
Same tree param. space	MotherTree	$\mathbf{0.692}_{\pm 0.104}$	$0.698_{\pm 0.133}$	$\underline{0.766}_{\pm 0.120}$	$0.764_{\pm 0.103}$
Same tree param. space	MotherTree-Tune	$\underline{0.680}_{\pm 0.101}$	$0.757_{\pm 0.166}$	$\mathbf{0.772}_{\pm 0.125}$	$\mathbf{0.804}_{\pm 0.117}$

13 multiclass) as used in MotherNet (Müller et al., 2025), and (ii) 38 data sets from the TabArena benchmark (Erickson et al., 2026) (30 binary, 8 multiclass). The benchmarks overlap on 4 tasks; when aggregating over both, we keep one of each duplicated task. See Appendix A.4 for details.

MotherTree learners are competitive across tasks and sample budgets. Figure 2 illustrates the performance of all methods for different training set budgets. For the smallest budget, $B = 100$, MotherTree and MotherTree-Tune achieve the best results on both TabArena and OpenML-CC18. This demonstrates that meta-learning can benefit tree fitting both directly (MotherTree) and as an initialization for subsequent optimization (MotherTree-Tune). CART is a consistently strong baseline (Figure 3), outperforming both optimal and gradient-based tree baselines at larger training set sizes. Only MotherTree-Tune remains competitive in this regime. GradTree is the strongest gradient-based baseline, and benefits substantially from distillation, but is consistently outperformed by MotherTree-Tune (we expand on this below). Appendix Table 8 shows the average fitting and inference time for evaluation tasks on both GPU and CPU. The time for MotherTree-Tune increases moderately with training set budget, remaining within a few seconds per task on both GPU and CPU.

Meta-learning with MotherTree outperforms from-scratch gradient-based learning. A close comparison with GradTree reveals the utility of meta-learning: Base MotherTree optimizes the same parameterization as GradTree using a single forward pass through the meta-learned generator, rather than from-scratch stochastic gradient descent. From Figure 2, MotherTree outperforms GradTree up to a budget of $B = 800$ samples on both benchmarks, with the methods performing comparably on



(a) Training set budget $B = 100$ (b) Training set budget $B = 1000$

Figure 3: Critical Difference Diagram across both benchmarks at different training set sample budgets, B. More comparisons across other context budgets are reported in Appendix A.7

OpenML-CC18 at 1000 samples. Table 1 separates results for binary and multiclass classification tasks at $B = 200$ samples. In binary tasks, MotherTree improves the mean AUC in binary tasks over GradTree from 0.635 to 0.692 on TabArena and from 0.725 to 0.766 on OpenML-CC18-Small. MotherTree also outperform other gradient-based baselines at this budget, improving over RADDT by 0.065 (0.056) AUC and over DNDT by 0.097 (0.116) points on TabArena and (OpenML-CC18), respectively. For multiclass tasks, MotherTree improves over GradTree on OpenML-CC18 but underperforms on TabArena, indicating less consistent gains on this benchmark. RADDT is best among gradient-based methods at multiclass tasks with $B = 200$, but *returns an oblique (not axis-aligned) tree*. The gap between MotherTree and GradTree decreases as the training data size grows, suggesting that the meta-learned prior is most beneficial when task-specific observations are scarce; fitting a tree from scratch using gradient descent becomes increasingly competitive with more data.

Meta-learning and task-specific optimization are complementary. MotherTree-Tune continues to optimize the parameters generated by MotherTree using the same procedure and hyperparameters as GradTree. Notably, MotherTree-Tune consistently outperforms GradTree and GradTree$_{\leftarrow \mathrm{TabPFN}}$ across training set budgets (Figures 2 & 3), benchmarks, and label types (Table 1), demonstrating the value of MotherTree as an initializer for gradient-based learning. Task-specific tuning is often preferable over base MotherTree: At $B = 200$, on binary TabArena tasks, MotherTree-Tune decreases AUC slightly from 0.692 to 0.680, but improves it from 0.766 to 0.772 on OpenML-CC18. The benefit is more pronounced for multiclass tasks, where tuning improves MotherTree from 0.698 to 0.757 on TabArena and from 0.764 to 0.804 on OpenML-CC18, outperforming the more flexible parameterization of RADDT. These results suggest that *meta-learning and task-specific optimization are complementary:* MotherTree provides a task-dependent initialization that i) consistently outperforms from-scratch optimization, and ii) can be improved further by continued tuning.

Parameter meta-learning is an effective mechanism for transferring prior knowledge to trees. Finally, we compare different mechanisms for transferring prior knowledge to decision trees. These methods differ substantially in how the prior is represented and transferred. Our distillation baselines transfer the predictive knowledge of TabPFN2.6 separately on each downstream task through teacher-generated labels. MetaTree meta-learns from pre-fit reference trees, thereby transferring an explicit prior over tree structures, whereas LLM-Tree constructs trees from the semantic information associated with table columns. In contrast, MotherTree meta-learns the tree-generation procedure from synthetic tasks and conditions the resulting structural prior directly on the observed context.

Figure 3 reveals a clear dependence on the amount of task-specific data. At a training budget of $B = 100$, MotherTree-Tune achieves the best average rank across the joint benchmark, while MotherTree also ranks among the strongest methods. In contrast, distillation provides limited benefit in this regime: both CART$_{\leftarrow \mathrm{TabPFN}}$ and GradTree$_{\leftarrow \mathrm{TabPFN}}$ rank below their meta-learned counterparts. As the context budget increases to $B = 1000$, the gap narrows substantially, and CART$_{\leftarrow \mathrm{TabPFN}}$ becomes the highest-ranked method. This behavior is consistent with their respective transfer mechanisms: MetaTree learns from shallow target trees, while LLM-Tree primarily relies on table semantics rather than increasing amounts of labeled context.

Ablation studies: Tree-aware decoder. We ablate the tree-structured Transformer decoder by replacing it with the MLP decoder used in MotherNet, modifying only its output dimension to match

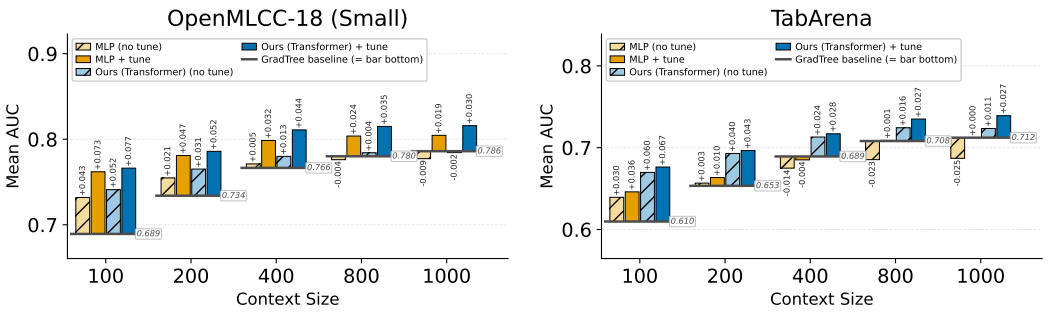


Figure 4: Decoder ablation on OpenML-CC18-Small and TabArena. Bars show the change in mean ROC-AUC relative to GradTree at each context size. We compare the MotherNet-style MLP decoder with our tree-structured Transformer decoder, both before and after task-specific finetuning.

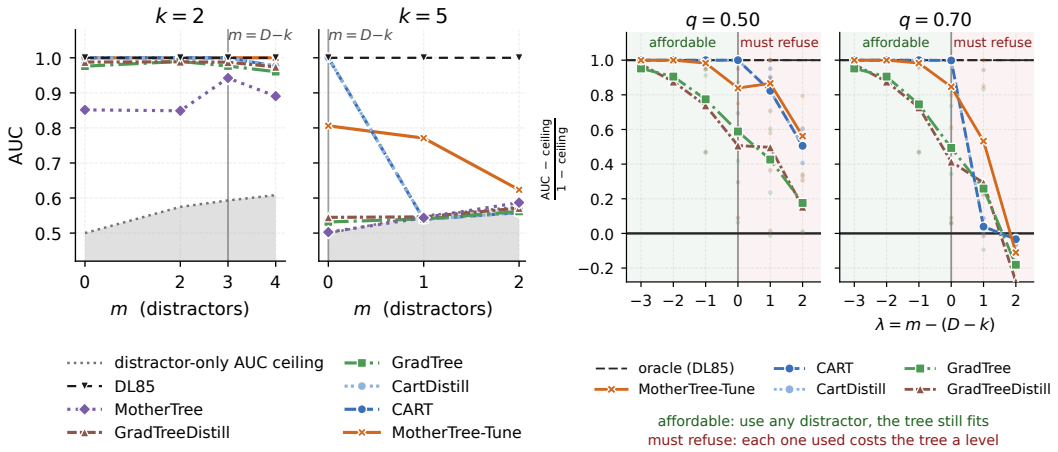
the parameters of the downstream tree. Figure 4 shows that the Transformer decoder consistently provides a stronger initialization than the MLP decoder. The difference is particularly clear after task-specific tuning: Transformer-decoded trees outperform GradTree across all context sizes on both benchmarks, whereas the advantage of the MLP initialization diminishes as more training data are added. These results suggest that explicitly modeling dependencies among tree nodes provides a more effective inductive bias for generating and initializing decision trees. In Appendix A.2, we compare using the MotherNet and TabICLv2 encoders and find no benefit from the latter.

4.2 Introspection on synthetic parity (XOR) tasks

Real-world benchmarks do not reveal whether performance gains arise from recovering more appropriate tree structures, since the optimal splits are unknown. To test whether meta-learned tree induction improves *feature selection*, we construct synthetic binary classification tasks using parity (XOR) functions corrupted with individually-informative distractors. Every distractor is informative about the label, whereas every parity bit is informative only through its joint interaction. Thus, greedy splitting is non-optimal by design. Given k bits $\mathbf{b}$, a noiseless label is defined as $y = \bigoplus_{i=1}^{k} b_i$, where b_i is drawn uniformly at random. We then form the input $\mathbf{x}$ by appending m distractor features to $\mathbf{b}$, each of which is a noisy copy of the label: distractor j equals y with probability q and is flipped with $1 - q$ probability. We compare tree learners to the "ceiling" achievable by a distractor-only model. For details about the Parity tasks, see Appendix A.8.1.

Higher-order parity is challenging for both gradient-based and greedy tree optimization. Figure 5a first shows that the difficulty of the task is not caused solely by distractor features. For larger parity orders ($k = 5$), gradient-based tree methods already exhibit a gap to the DL85 oracle and greedy splitting (CART) when no distractors are present ($m = 0$). This reflects the difficulty of *optimizing* the differentiable tree parameterization to fit higher-order interactions, rather than an intrinsic limitation of the tree structure itself or sample inefficiency (since the label is noiseless). We therefore focus on the additional degradation caused by distractors, rather than interpreting the absolute gap to the oracle as feature-selection error.

Meta-learned initialization improves robustness under a tight depth budget. The effect of distractors becomes pronounced once $\lambda := m - (D - k) > 0$. In this regime, the depth budget is exhausted, so using a distractor consumes a split that is required to fully represent the parity function. Most tree learners deteriorate sharply as λ increases. In contrast, MotherTree-Tune degrades more gradually and remains substantially more robust across the difficult configurations, indicating that the meta-learned initialization helps task-specific optimization resist locally predictive but structurally sub-optimal features. Figure 5b isolates this effect across different distractor strengths q by normalizing performance relative to the distractor-only "ceiling" and the DL85 oracle. Across values of q, MotherTree-Tune retains a larger fraction of the attainable improvement once the depth constraint requires distractors to be rejected. These results suggest that the main benefit of meta-learning in this setting is not to solve the parity problem outright, but to provide a stronger initialization that makes gradient-based tree optimization more robust to misleading shortcut features.



(a) AUC against the number of distractor features m for varying parity orders k (number of bits).

(b) Realized improvement vs optimal distractor-only model under different distractor-label correlation, q.

Figure 5: Learning parity with distractors at fixed maximum tree depth, $D = 5$. Right: $\lambda = m - (D - k)$ is the number of distractors that must be ignored to learn an optimal tree. Green ($\lambda \leq 0$): the depth budget has slack and splitting on a distractor does not rule out an optimal model. Red ($\lambda > 0$): the budget is exhausted, so each distractor used takes a level away from the parity tree; at least λ of them must be refused. Results across all settings could be seen in Appendix A.8.3.

5 Discussion

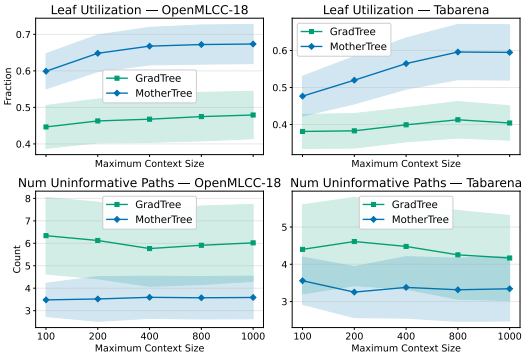


Figure 6: Number of uninformative paths (lower is better) and leaf utilization (higher is better).

When does amortized tree learning help? In our experiments, MotherTree provides the largest gains when task-specific training sets are limited in size; the advantage over fitting the same tree parameterization from scratch decreases as more data become available. This is consistent with the role of meta-learning as a task-conditioned prior: it provides a stronger inductive bias when observations are scarce, whereas task-specific optimization becomes increasingly competitive with more data. Currently, the gains are strongest on binary classification tasks, while the weaker multiclass performance suggests that direct generation also inherits limitations of the underlying differentiable tree parameterization. Importantly, the structural statistics from Figure 6 show that this prior affects the tree itself: MotherTree uses a larger fraction of leaves and produces fewer contradictory root-to-leaf paths than GradTree under the same depth and parameterization, which shows that the improvement from meta-learning also leads to a better tree structure.

Limitations. MotherTree currently generates complete binary trees with a fixed topology and depth. This simplifies amortized generation, since the number and correspondence of tree parameters are known in advance, but prevents the model from adapting tree size or topology to individual tasks. As a result, some generated trees may allocate capacity to unnecessary nodes, while more complex tasks may require a larger tree than the predefined structure allows. Extending MotherTree to generate variable-size or pruned tree structures is an important direction for future work.

Our study demonstrates the value of meta-learning induction of *single-tree* classifiers. However, tree *ensembles* universally achieve stronger predictive performance by combining predictions of multiple complementary trees. Future research extending amortized tree generation to ensembles must tackle questions about how to encourage diversity and coordinate dependency structures across generated trees, to minimize redundancy and improve coverage of the target task.

AI USE STATEMENT

In this work, we used generative AI tools to polish language, to provide feedback on the clarity and consistency of the presentation, and to verify the alignment between our methodological descriptions and code implementation. We also used generative AI tools to help navigate, retrieve, and understand relevant code from external repositories. Generative AI tools were not used to implement the proposed models, training pipelines, or data set-generation procedures, nor to generate or alter experimental results. Additionally, we used generative AI tools to assist with the creation and modification of figures. All AI-assisted figures were manually reviewed, and their underlying source data were checked against the corresponding experimental results. We reviewed all AI-assisted content and take full responsibility for the final manuscript, including its text, claims, code-related descriptions, figures, and other artifacts produced with the aid of generative AI.

ETHICS STATEMENT

No new data collection was performed during this study, nor were any studies performed on human subjects. We don't anticipate any harm directly attributable to the methods developed in this work.

REPRODUCIBILITY STATEMENT

All data sets used in this work are either already public or described in the Appendix of this paper, in the case of the synthetic parity task. Existing, public implementations of baseline algorithms were used where possible and hyperparameter settings were reported in the Appendix. The implementations of all experiments and our novel models were written in Python for general-purpose hardware and will be released in a public code repository at the first de-anonymized release of this work, along with instructions and scripts to reproduce the figures and tables in the submission.

REFERENCES

Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 3146–3153, 2020.

Maria Florina Balcan and Dravyansh Sharma. Learning accurate and interpretable decision trees. In *The 40th Conference on Uncertainty in Artificial Intelligence*, 2024.

Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Pieter Gijsbers, Frank Hutter, Michel Lang, Rafael G Mantovani, Jan N van Rijn, and Joaquin Vanschoren. Openml benchmarking suites. *arXiv preprint arXiv:1708.03731*, 2017.

Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Vinícius G Costa and Carlos E Pedreira. Recent advances in decision trees: an updated survey: Vg costa, ce pedreira. *Artificial Intelligence Review*, 56(5):4765–4800, 2023.

Nick Erickson, Lennart Purucker, Andrej Tschalzev, David Holzmüller, Prateek Desai, David Salinas, and Frank Hutter. Tabarena: A living benchmark for machine learning on tabular data. *Advances in Neural Information Processing Systems*, 38, 2026.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

Léo Grinsztajn, Klemens Flöge, Oscar Key, Felix Birkel, Philipp Jund, Brendan Roof, Mihir Manium, Shi Bin Hoo, Magnus Bühler, Anurag Garg, et al. Tabpfn-3: Technical report. *arXiv preprint arXiv:2605.13986*, 2026.

David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks, 2016.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations*, 2023.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025.

David Holzmüller, Leo Grinsztajn, and Ingo Steinwart. Better by default: Strong pre-tuned MLPs and boosted trees on tabular data. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. *Advances in neural information processing systems*, 32, 2019.

Ozan Irsoy, Olcay Taner Yıldız, and Ethem Alpaydın. Soft decision trees. In *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*, pp. 1819–1822. IEEE, 2012.

Ricardo Knauer, Mario Koddenbrock, Raphael Wallsberger, Nicholas M Brisson, Georg N Duda, Deborah Falla, David W Evans, and Erik Rodner. 'oh llm, i'm asking thee, please give me a decision tree': Zero-shot decision tree induction and embedding with large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 1196–1206, 2025.

Haoling Li, Jie Song, Mengqi Xue, Haofei Zhang, Jingwen Ye, Lechao Cheng, and Mingli Song. A survey of neural trees. *arXiv preprint arXiv:2209.03415*, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Mohammed Mahfoud, Ghait Boukachab, MichaĹ Koziarski, Alex Hernandez-Garcia, Stefan Bauer, Yoshua Bengio, and Nikolay Malkin. Learning decision trees as amortized structure inference. *arXiv preprint arXiv:2503.06985*, 2025.

Qiangqiang Mao, Jiayang Ren, Yixiu Wang, Chenxuanyin Zou, Jingjing Zheng, and Yankai Cao. Differentiable decision tree via "relu+argmin" reformulation. In *Advances in Neural Information Processing Systems*, 2025.

Sascha Marton, Stefan Lüdtke, Christian Bartelt, and Heiner Stuckenschmidt. Gradtree: Learning axis-aligned decision trees with gradient descent. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 14323–14331, 2024.

Andreas Müller, Carlo Curino, and Raghu Ramakrishnan. Mothernet: Fast training and inference via hyper-network transformers. In *International Conference on Learning Representations*, volume 2025, pp. 76666–76686, 2025.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *International Conference on Learning Representations*, 2022.

Mohammad Norouzi, Maxwell Collins, Matthew A Johnson, David J Fleet, and Pushmeet Kohli. Efficient non-greedy optimization of decision trees. *Advances in neural information processing systems*, 28, 2015.

Alizée Pace, Alex J Chan, and Mihaela van der Schaar. Poetree: Interpretable policy learning with adaptive decision trees. *arXiv preprint arXiv:2203.08057*, 2022.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Lennart Purucker, Andrej Tschalzev, Nick Erickson, Gioia Blayer, David HolzmÃ1/4ller, Alan Arazi, Alexander Pfefferle, Mustafa Tajjar, GaÃ Varoquaux, Frank Hutter, et al. Beyond iid: How general are tabular foundation models, really? *arXiv preprint arXiv:2606.30410*, 2026.

Jingang QU, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A tabular foundation model for in-context learning on large data. In *Forty-second International Conference on Machine Learning*, 2025.

Jingang QU, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICLv2: A better, faster, scalable, and open tabular foundation model. In *Forty-third International Conference on Machine Learning*, 2026.

John Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc, 1993. ISBN 1-55860-238-0.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International conference on learning representations*, 2017.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5):206–215, 2019.

Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistic Surveys*, 16:1–85, 2022.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

scikit-learn. scikit-learn User Guide 1.10.6. Tree algorithms: ID3, C4.5, C5.0 and CART. `https://scikit-learn.org/stable/modules/tree.html#tree-algorithms`, 2026. Accessed: 2026-09-10.

Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In *International Conference on Machine Learning*, pp. 6166–6175. PMLR, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Yongxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. Deep neural decision trees. *arXiv preprint arXiv:1806.06988*, 2018.

Yufan Zhuang, Liyuan Liu, Chandan Singh, Jingbo Shang, and Jianfeng Gao. Learning a decision tree algorithm with transformers. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

A Appendix

A.1 Architecture details

Encoder We keep the same encoder from MotherNet (Müller et al., 2025), where each row of the table is embedded through a linear layer and one-hot encoding for class is used before sending to the transformer blocks. Unlike TabPFN (Hollmann et al., 2025) and TabICL (QU et al., 2026), which rely on dual attention between context and features, MotherNet encoder operates exclusively on the context data and does not require test points to participate in attention.

Concretely, given a context set with n samples and d features, the encoder first applies a linear projection to map each row and corresponding label into a d'-dimensional feature embedding. Addition is used to combine feature and label information, followed by 12 inter-sample attention layers to capture the mutual information between rows. This yields a row-wise representation for every context sample. To aggregate the n row representations into a single data set embedding, the encoder performs class-wise mean pooling: it averages the row representations belonging to the same class, and concatenates the resulting k class means into $\mathrm{E}_{\phi_{\mathrm{enc}}}(\mathcal{D})$. The resulting $\mathrm{E}_{\phi_{\mathrm{enc}}}(\mathcal{D})$ is fixed-dimensional, which the tree decoder subsequently maps to the parameters of an axis-aligned decision tree.

Decoder For node representations $H^{(i)} \in \mathbb{R}^{N \times d_n}$, ancestral self-attention is computed as

$$\mathbf{Q} = H^{(i)}\mathbf{W}_Q, \quad \mathbf{K} = H^{(i)}\mathbf{W}_K, \quad \mathbf{V} = H^{(i)}\mathbf{W}_V,$$

$$\mathrm{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{h}} + \mathbf{M}\right)\mathbf{V}.$$

After L Transformer layers, the resulting node representations are projected to the corresponding split and leaf parameters, yielding $\theta = (I, T, L)$. We set $d_n = 32$ and use $L = 8$ Transformer layers, resulting in a decoder with 23.6M parameters. These hyperparameters were chosen to keep the decoder capacity comparable to that of the original MotherNet decoder (17.7M parameters) for tree generation.

A.2 Comparison with TabICL encoder

We also ablate the MotherTree encoder by replacing the MotherNet encoder with the encoder from TabICLv2 (QU et al., 2026). We provide only the training samples to the encoder and use the same aggregation procedure to construct the data set embedding, as described in Section 3.2.

Table 2: Mean AUC for different encoders on TabArena.

Context	TabICL Encoder	MotherNet Encoder
100	0.632	0.639
200	0.652	0.657
400	0.671	0.675
800	0.679	0.685
1000	0.681	0.687

As shown in Table 2, replacing the MotherNet encoder with TabICLv2 does not improve performance across the evaluated context sizes. In fact, the MotherNet encoder performs slightly better in all settings. These results suggest that the performance of MotherTree is not driven by the use of a stronger tabular prediction backbone, and that increased encoder sophistication does not necessarily translate into better representations for tree generation.

A.3 Training Details

MotherTree is pre-trained on both TabICL (QU et al., 2025) and TabICLv2 prior (QU et al., 2026). The main reason is that, for TabICL prior, it provides diverse activation functions and tree-based SCMs to model hierarchical interactions between variables, which helps the model to grasp the basic complicated interaction between variables. However, due to the instability of some activation functions like `Exponential function`, it sometimes introduces the overflow error, which makes the data set quality less effective. While in TabICLv2, a more sophisticated prior generation

process is utilized. `Random Cauchy graph` creates richer connectivity of graph, but the generation process is much more expensive than the previous one. Hence, during the first stage of training, our model is trained on the fly with generation of 128 batches of data for each step. A LinearLR scheduler is used to gradually increase the learning rate. For the second stage of training, we generate $120k$ files on the disk. For each file, it contains 4 groups of graph configuration. For each configuration, we randomly generate 16 batches. 1000 steps of warm up is firstly applied and then we switch to cosine annealing scheduler by setting minimum learning rate to 1×10^{-8}. We optimize the MotherTree using the AdamW optimizer (Loshchilov & Hutter, 2017) and the momentum coefficients are set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

As shown in Table 3, incorporating Stage 1 pre-training with the TabICL prior substantially improves the validation AUC compared with training only on the TabICLv2 prior. This design is also considerably more training-efficient, since generating TabICLv2 prior samples is substantially more expensive and would otherwise lead to a much longer overall training time. For training hours, Stage 1 took 68 hours and stage 2 took 18 hours.

Table 3: Impact of Stage 1 pre-training($400k$ steps) on validation sets from OpenML-CC18 (Müller et al., 2025) after $125k$ steps.

Training Scheme	Mean AUC
Stage 2 only	0.685
Stage 1 + Stage 2	0.752

A.4 Evaluation Details

A.4.1 Duplicate Tasks

Between TabArena and OpenMLCC-18, there are four overlapping data sets, which are `blood-transfusion-service-center`, `diabetes`, `credit-g` and `qsar-biodeg`. For the first two data sets, they are exactly the same data sets but orders are different. For `credit-g`, the original ordinal encoding is reversed, which doesn't affect the results. `qsar-biodeg` is the one case where the two copies are not the same size(TabArena has one less row) and TabArena changes some of numeric-ordinal features to categorical features.

A.4.2 Distillation Baseline

Following MotherNet (Müller et al., 2025), we also incorporate distillation baselines for comparison. Distillation provides a natural way to transfer knowledge from a powerful teacher model (e.g., TabPFN) to a lightweight student model. However, unlike the original distillation framework (Hinton et al., 2015), which relies on gradient descent with soft labels, the non-differentiability of CART prevents direct application of this approach. We therefore adopt a simplified strategy: hard labels sampled from the predicted probability of the teacher model are mixed with the original training data, and the student model is trained on this augmented data set.

For consistency and simplicity, we apply this strategy to both CART and GradTree. Since both MotherTree and these distilled baselines leverage knowledge from PFN(prior fitted network), any performance gap between them can be attributed primarily to architectural differences rather than the source of knowledge. We sample the label for each training data according to the predictions from the teacher model. Then, append the sample with new label to the training set, which doubles the original size of the training set.

As for the teacher model, we experiment with both `TabPFN2.6` and `TabPFN-3` (Grinsztajn et al., 2026). We report results from `TabPFN2.6` in the main paper, as we observe no significant performance difference between the two teachers, while `TabPFN2.6` is substantially more time-efficient.

A.4.3 Baselines & Hyperparameter tuning

We compare our method against several baseline models using their publicly available implementations. The corresponding repositories are listed below:

- DNDT: `https://github.com/wOOL/DNDT`
- RADDT: `https://github.com/YankaiGroup/RADDT`
- LLM-Tree: `https://github.com/ml-lab-htw/llm-trees`

- GradTree (we re-implemented the following repository in `Pytorch`): `https://github.com/s-marton/GradTree`
- DL85: `https://github.com/aia-uclouvain/pydl8.5`
- MetaTree: `https://github.com/EvanZhuang/MetaTree`

Table 4: Hyperparameter search spaces and fixed settings for each model.

Model	Hyperparameters
CART	criterion $\in$ {gini, entropy}; min_samples_split $\in \{2, 5, 10\}$; min_samples_leaf $\in \{1, 2, 5\}$.
GradTree	feature selector LR $\in \{0.001, 0.005, 0.010, 0.050\}$; threshold LR $\in \{0.010, 0.050, 0.100, 0.250\}$; leaf node LR $\in \{0.001, 0.005, 0.010, 0.100\}$; epochs $= 1000$; patience $= 50$.
DL85	time limit $= 600$s; max depth $= 5$; quantile binning $= 5$.
(α^*, β^*) Tsallis	$a \in \text{linspace}(1, 10, 10)$; $b \in \text{linspace}(1, 10, 10)$.
DNDT	learning rate $= 0.05$; batch size $= 256$; epochs $= 300$; patience $= 20$; max features $= 10$; Gumbel temperature $1.0 \to 0.1$ over 200 epochs.
RADDT	tree depth $= 5$; epochs $= 300$; num scale $= 5$; start num $= 1$; warm start: CART (1 initialization).
MetaTree	tree depth $= 5$; max features $= 10$; max samples $= 256$.
LLM-Tree	DeepSeek-V4-Flash (thinking disabled).

A.4.4 SAMPLE METHOD

Algorithm 1 Deterministic Valid Train–Evaluation Split

Require: Data set (X, y), sample size n, training size k, seed s, minimum class count m
Ensure: Ordered split (X', y')

$n' \leftarrow \min(n, |X|)$
Remove missing samples and classes with fewer than two samples
if $n < |X|$ **then**
  $(\mathcal{C}, \mathbf{c}) \leftarrow$ UNIQUEWITHCOUNTS(y)
  Initialize $\mathbf{a} \leftarrow \mathbf{0}$
  ▷ Minimum allocation
  for c_i in ascending order of c_i **do**
    $a_i \leftarrow \min\{m, c_i, n - \sum_j a_j\}$
  end for
  ▷ Proportional allocation of remaining slots
  $r \leftarrow n - \sum_i a_i$
  $q_i \leftarrow r \frac{c_i - a_i}{\sum_j (c_j - a_j)}$
  $e_i \leftarrow \lfloor q_i \rfloor$
  $\ell \leftarrow r - \sum_i e_i$
  Let $\mathcal{I}$ be the ℓ classes with largest $(q_i - e_i)$ among those with $e_i < c_i - a_i$
  $e_i \leftarrow e_i + 1$ for $i \in \mathcal{I}$
  $a_i \leftarrow a_i + e_i$
  Sample a_i instances from each class c_i
  $(X_s, y_s) \leftarrow$ sampled data set
else
  $(X_s, y_s) \leftarrow (X, y)$
end if
$(X_{\text{tr}}, X_{\text{te}}, y_{\text{tr}}, y_{\text{te}}) \leftarrow$ STRATIFIEDSPLIT(X_s, y_s, k, s)
Ensure both splits contain all classes by exchanging samples if necessary
return $(X_{\text{tr}}, X_{\text{te}}, y_{\text{tr}}, y_{\text{te}})$

A.4.5 PREPROCESSING PIPELINE AND MISSING VALUE HANDLING

In the original implementation of GradTree, categorical features are processed according to their cardinality. Features with fewer than 10 unique categories are transformed using a **One-Hot Encoder**, whereas categorical features with higher cardinality are transformed using a **Leave-One-Out (LOO) Encoder**. Based on our empirical experiments, however, we use the **Leave-One-Out Encoder for all categorical features**, regardless of their cardinality. This modification provides a unified representation for categorical variables while avoiding the potentially substantial increase in dimensionality introduced by one-hot encoding. Missing values are handled after categorical encoding using median imputation. Specifically, the median of each transformed feature is computed from the training data and used to replace remaining missing values. Following categorical encoding and missing-value imputation, we retain the **Quantile Transformer** used in the original GradTree implementation and apply it to all transformed features, including both the original numerical features and the LOO-encoded categorical features.

A.5 RESULTS ACORSS ALL LENGHTS

Table 5: Overall mean AUC across benchmarks — model comparison per budget.

Benchmark		Model	100	200	400	800	1000
TabArena		CART	$0.668_{\pm 0.121}$	$0.693_{\pm 0.128}$	$\mathbf{0.720}_{\pm 0.131}$	$\underline{0.743}_{\pm 0.130}$	$\underline{0.751}_{\pm 0.126}$
		CART$_{\leftarrow \text{TabPFN}}$	$0.655_{\pm 0.114}$	$0.684_{\pm 0.124}$	$0.715_{\pm 0.132}$	$\mathbf{0.751}_{\pm 0.128}$	$\mathbf{0.756}_{\pm 0.126}$
		(α^*, β^*) Tsallis	$0.645_{\pm 0.123}$	$0.650_{\pm 0.133}$	$0.655_{\pm 0.141}$	$0.654_{\pm 0.141}$	$0.654_{\pm 0.141}$
		DNDT	$0.629_{\pm 0.089}$	$0.647_{\pm 0.094}$	$0.659_{\pm 0.096}$	$0.665_{\pm 0.094}$	$0.662_{\pm 0.098}$
		RADDT	$0.642_{\pm 0.116}$	$0.651_{\pm 0.121}$	$0.670_{\pm 0.125}$	$0.686_{\pm 0.132}$	$0.688_{\pm 0.135}$
		DL85	$0.611_{\pm 0.089}$	$0.629_{\pm 0.111}$	$0.659_{\pm 0.124}$	$0.708_{\pm 0.126}$	$0.714_{\pm 0.127}$
		LLM-Tree	$0.630_{\pm 0.090}$	$0.644_{\pm 0.094}$	$0.662_{\pm 0.096}$	$0.673_{\pm 0.096}$	$0.679_{\pm 0.097}$
		MetaTree	$0.633_{\pm 0.100}$	$0.646_{\pm 0.102}$	$0.647_{\pm 0.106}$	$0.639_{\pm 0.104}$	$0.643_{\pm 0.105}$
	Same tree param. space	GradTree	$0.610_{\pm 0.083}$	$0.653_{\pm 0.101}$	$0.689_{\pm 0.117}$	$0.708_{\pm 0.111}$	$0.712_{\pm 0.116}$
		GradTree$_{\leftarrow \text{TabPFN}}$	$0.641_{\pm 0.101}$	$0.667_{\pm 0.108}$	$0.701_{\pm 0.119}$	$0.726_{\pm 0.116}$	$0.722_{\pm 0.119}$
		MotherTree	$\mathbf{0.679}_{\pm 0.107}$	$\mathbf{0.696}_{\pm 0.107}$	$0.705_{\pm 0.114}$	$0.715_{\pm 0.107}$	$0.716_{\pm 0.110}$
		MotherTree-Tune	$0.676_{\pm 0.110}$	$0.696_{\pm 0.119}$	$\underline{0.717}_{\pm 0.124}$	$0.735_{\pm 0.117}$	$0.739_{\pm 0.120}$
OpenML-CC18-Small		CART	$0.734_{\pm 0.120}$	$\underline{0.777}_{\pm 0.117}$	$\underline{0.805}_{\pm 0.120}$	$0.811_{\pm 0.123}$	$0.813_{\pm 0.124}$
		CART$_{\leftarrow \text{TabPFN}}$	$0.731_{\pm 0.111}$	$0.768_{\pm 0.117}$	$0.802_{\pm 0.117}$	$\underline{0.813}_{\pm 0.117}$	$\underline{0.815}_{\pm 0.118}$
		(α^*, β^*) Tsallis	$0.717_{\pm 0.124}$	$0.723_{\pm 0.146}$	$0.724_{\pm 0.163}$	$0.724_{\pm 0.167}$	$0.725_{\pm 0.168}$
		DNDT	$0.585_{\pm 0.066}$	$0.595_{\pm 0.074}$	$0.607_{\pm 0.076}$	$0.616_{\pm 0.082}$	$0.615_{\pm 0.085}$
		RADDT	$0.715_{\pm 0.132}$	$0.748_{\pm 0.136}$	$0.768_{\pm 0.145}$	$0.776_{\pm 0.146}$	$0.777_{\pm 0.149}$
		DL85	$0.670_{\pm 0.113}$	$0.713_{\pm 0.121}$	$0.747_{\pm 0.136}$	$0.761_{\pm 0.139}$	$0.762_{\pm 0.140}$
		LLM-Tree	$0.684_{\pm 0.135}$	$0.699_{\pm 0.136}$	$0.715_{\pm 0.135}$	$0.718_{\pm 0.137}$	$0.718_{\pm 0.137}$
		MetaTree	$0.691_{\pm 0.127}$	$0.716_{\pm 0.124}$	$0.726_{\pm 0.123}$	$0.721_{\pm 0.121}$	$0.723_{\pm 0.121}$
	Same tree param. space	GradTree	$0.689_{\pm 0.123}$	$0.734_{\pm 0.124}$	$0.766_{\pm 0.122}$	$0.780_{\pm 0.122}$	$0.786_{\pm 0.122}$
		GradTree$_{\leftarrow \text{TabPFN}}$	$0.719_{\pm 0.121}$	$0.762_{\pm 0.117}$	$0.784_{\pm 0.121}$	$0.795_{\pm 0.119}$	$0.799_{\pm 0.119}$
		MotherTree	$\underline{0.741}_{\pm 0.111}$	$0.765_{\pm 0.111}$	$0.780_{\pm 0.114}$	$0.784_{\pm 0.114}$	$0.784_{\pm 0.114}$
		MotherTree-Tune	$\mathbf{0.766}_{\pm 0.117}$	$\mathbf{0.786}_{\pm 0.121}$	$\mathbf{0.811}_{\pm 0.113}$	$\mathbf{0.815}_{\pm 0.116}$	$\mathbf{0.816}_{\pm 0.117}$

Table 6: Mean AUC for binary classification tasks on TabArena and OpenML-CC18

Benchmark		Model	100	200	400	800	1000
TabArena		CART	$0.647_{\pm 0.101}$	$0.670_{\pm 0.107}$	$0.697_{\pm 0.106}$	$0.722_{\pm 0.108}$	$\underline{0.731}_{\pm 0.104}$
		$\text{CART}_{\leftarrow \text{TabPFN}}$	$0.634_{\pm 0.092}$	$0.659_{\pm 0.099}$	$0.690_{\pm 0.108}$	$\mathbf{0.732}_{\pm 0.107}$	$\mathbf{0.738}_{\pm 0.104}$
		(α^*, β^*) Tsallis	$0.619_{\pm 0.100}$	$0.620_{\pm 0.108}$	$0.624_{\pm 0.118}$	$0.623_{\pm 0.116}$	$0.623_{\pm 0.117}$
		DNDT	$0.586_{\pm 0.071}$	$0.595_{\pm 0.079}$	$0.607_{\pm 0.079}$	$0.614_{\pm 0.088}$	$0.616_{\pm 0.089}$
		RADDT	$0.620_{\pm 0.095}$	$0.627_{\pm 0.100}$	$0.646_{\pm 0.108}$	$0.660_{\pm 0.115}$	$0.662_{\pm 0.114}$
		DL85	$0.594_{\pm 0.073}$	$0.604_{\pm 0.088}$	$0.631_{\pm 0.096}$	$0.685_{\pm 0.103}$	$0.692_{\pm 0.106}$
		MetaTree	$0.614_{\pm 0.076}$	$0.632_{\pm 0.085}$	$0.640_{\pm 0.090}$	$0.631_{\pm 0.087}$	$0.633_{\pm 0.092}$
		LLM-Tree	$0.629_{\pm 0.091}$	$0.641_{\pm 0.092}$	$0.661_{\pm 0.094}$	$0.671_{\pm 0.095}$	$0.679_{\pm 0.096}$
	Same tree param. space	GradTree	$0.604_{\pm 0.082}$	$0.635_{\pm 0.081}$	$0.674_{\pm 0.101}$	$0.695_{\pm 0.097}$	$0.700_{\pm 0.102}$
	Same tree param. space	$\text{GradTree}_{\leftarrow \text{TabPFN}}$	$0.629_{\pm 0.093}$	$0.655_{\pm 0.095}$	$0.684_{\pm 0.103}$	$0.715_{\pm 0.102}$	$0.711_{\pm 0.104}$
	Same tree param. space	MotherTree	$\mathbf{0.668}_{\pm 0.105}$	$\mathbf{0.692}_{\pm 0.104}$	$\mathbf{0.712}_{\pm 0.106}$	$\underline{0.724}_{\pm 0.103}$	$0.722_{\pm 0.100}$
	Same tree param. space	MotherTree-Tune	$\underline{0.663}_{\pm 0.092}$	$\underline{0.680}_{\pm 0.101}$	$\underline{0.701}_{\pm 0.105}$	$0.723_{\pm 0.098}$	$0.728_{\pm 0.104}$
OpenML-CC18-Small		CART	$0.709_{\pm 0.129}$	$0.759_{\pm 0.123}$	$\underline{0.780}_{\pm 0.122}$	$0.783_{\pm 0.123}$	$0.783_{\pm 0.123}$
		$\text{CART}_{\leftarrow \text{TabPFN}}$	$0.712_{\pm 0.120}$	$0.748_{\pm 0.125}$	$0.777_{\pm 0.120}$	$\underline{0.788}_{\pm 0.115}$	$\underline{0.788}_{\pm 0.115}$
		(α^*, β^*) Tsallis	$0.684_{\pm 0.133}$	$0.680_{\pm 0.155}$	$0.675_{\pm 0.164}$	$0.672_{\pm 0.161}$	$0.672_{\pm 0.161}$
		DNDT	$0.635_{\pm 0.106}$	$0.650_{\pm 0.108}$	$0.658_{\pm 0.109}$	$0.660_{\pm 0.107}$	$0.660_{\pm 0.109}$
		RADDT	$0.684_{\pm 0.136}$	$0.710_{\pm 0.135}$	$0.726_{\pm 0.144}$	$0.732_{\pm 0.143}$	$0.730_{\pm 0.143}$
		DL85	$0.657_{\pm 0.128}$	$0.691_{\pm 0.131}$	$0.711_{\pm 0.142}$	$0.721_{\pm 0.141}$	$0.721_{\pm 0.141}$
		MetaTree	$0.687_{\pm 0.145}$	$0.705_{\pm 0.141}$	$0.711_{\pm 0.141}$	$0.711_{\pm 0.141}$	$0.711_{\pm 0.141}$
		LLM-Tree	$0.659_{\pm 0.147}$	$0.669_{\pm 0.145}$	$0.686_{\pm 0.142}$	$0.687_{\pm 0.143}$	$0.687_{\pm 0.143}$
	Same tree param. space	GradTree	$0.705_{\pm 0.136}$	$0.725_{\pm 0.132}$	$0.752_{\pm 0.127}$	$0.765_{\pm 0.128}$	$0.766_{\pm 0.125}$
	Same tree param. space	$\text{GradTree}_{\leftarrow \text{TabPFN}}$	$0.708_{\pm 0.137}$	$0.752_{\pm 0.128}$	$0.770_{\pm 0.135}$	$0.780_{\pm 0.127}$	$0.785_{\pm 0.127}$
	Same tree param. space	MotherTree	$\underline{0.747}_{\pm 0.124}$	$\underline{0.766}_{\pm 0.120}$	$0.775_{\pm 0.124}$	$0.776_{\pm 0.124}$	$0.776_{\pm 0.124}$
	Same tree param. space	MotherTree-Tune	$\mathbf{0.759}_{\pm 0.124}$	$\mathbf{0.772}_{\pm 0.125}$	$\mathbf{0.796}_{\pm 0.117}$	$\mathbf{0.797}_{\pm 0.120}$	$\mathbf{0.797}_{\pm 0.120}$

Table 7: Mean AUC for multi-class tasks on TabArena and OpenML-CC18.

Benchmark		Model	100	200	400	800	1000
TabArena		CART	$\mathbf{0.748}_{\pm 0.161}$	$\underline{0.779}_{\pm 0.170}$	$\mathbf{0.808}_{\pm 0.180}$	$\underline{0.819}_{\pm 0.178}$	$\underline{0.824}_{\pm 0.176}$
		$\text{CART}_{\leftarrow \text{TabPFN}}$	$0.736_{\pm 0.155}$	$\mathbf{0.780}_{\pm 0.165}$	$\underline{0.806}_{\pm 0.177}$	$\mathbf{0.822}_{\pm 0.179}$	$\mathbf{0.826}_{\pm 0.180}$
		(α^*, β^*) Tsallis	$\underline{0.744}_{\pm 0.156}$	$0.762_{\pm 0.167}$	$0.769_{\pm 0.168}$	$0.772_{\pm 0.171}$	$0.771_{\pm 0.170}$
		DNDT	$0.578_{\pm 0.044}$	$0.595_{\pm 0.055}$	$0.605_{\pm 0.068}$	$0.622_{\pm 0.057}$	$0.608_{\pm 0.073}$
		RADDT	$0.724_{\pm 0.153}$	$0.744_{\pm 0.152}$	$0.760_{\pm 0.149}$	$0.780_{\pm 0.159}$	$0.785_{\pm 0.171}$
		DL85	$0.675_{\pm 0.117}$	$0.722_{\pm 0.142}$	$0.763_{\pm 0.164}$	$0.792_{\pm 0.174}$	$0.798_{\pm 0.170}$
		MetaTree	$0.703_{\pm 0.147}$	$0.696_{\pm 0.145}$	$0.673_{\pm 0.159}$	$0.670_{\pm 0.155}$	$0.681_{\pm 0.145}$
		LLM-Tree	$0.634_{\pm 0.096}$	$0.654_{\pm 0.105}$	$0.669_{\pm 0.109}$	$0.678_{\pm 0.108}$	$0.679_{\pm 0.107}$
	Same tree param. space	GradTree	$0.631_{\pm 0.091}$	$0.723_{\pm 0.141}$	$0.748_{\pm 0.157}$	$0.756_{\pm 0.153}$	$0.757_{\pm 0.159}$
	Same tree param. space	$\text{GradTree}_{\leftarrow \text{TabPFN}}$	$0.686_{\pm 0.124}$	$0.715_{\pm 0.146}$	$0.761_{\pm 0.161}$	$0.767_{\pm 0.160}$	$0.763_{\pm 0.165}$
	Same tree param. space	MotherTree	$0.675_{\pm 0.119}$	$0.698_{\pm 0.133}$	$0.716_{\pm 0.136}$	$0.726_{\pm 0.137}$	$0.727_{\pm 0.134}$
	Same tree param. space	MotherTree-Tune	$0.728_{\pm 0.159}$	$0.757_{\pm 0.166}$	$0.779_{\pm 0.172}$	$0.778_{\pm 0.174}$	$0.779_{\pm 0.170}$
OpenML-CC18-Small		CART	$\underline{0.767}_{\pm 0.104}$	$\underline{0.802}_{\pm 0.110}$	$\mathbf{0.836}_{\pm 0.115}$	$\mathbf{0.849}_{\pm 0.117}$	$\mathbf{0.853}_{\pm 0.119}$
		$\text{CART}_{\leftarrow \text{TabPFN}}$	$0.756_{\pm 0.098}$	$0.795_{\pm 0.105}$	$\underline{0.835}_{\pm 0.110}$	$\underline{0.846}_{\pm 0.115}$	$\underline{0.849}_{\pm 0.116}$
		(α^*, β^*) Tsallis	$0.760_{\pm 0.101}$	$0.780_{\pm 0.115}$	$0.789_{\pm 0.143}$	$0.792_{\pm 0.156}$	$0.793_{\pm 0.158}$
		DNDT	$0.620_{\pm 0.065}$	$0.643_{\pm 0.076}$	$0.660_{\pm 0.081}$	$0.671_{\pm 0.080}$	$0.666_{\pm 0.087}$
		RADDT	$0.755_{\pm 0.119}$	$0.798_{\pm 0.124}$	$0.822_{\pm 0.133}$	$0.833_{\pm 0.136}$	$0.837_{\pm 0.139}$
		DL85	$0.686_{\pm 0.093}$	$0.743_{\pm 0.104}$	$0.793_{\pm 0.115}$	$0.814_{\pm 0.123}$	$0.815_{\pm 0.124}$
		MetaTree	$0.697_{\pm 0.104}$	$0.731_{\pm 0.103}$	$0.744_{\pm 0.097}$	$0.734_{\pm 0.091}$	$0.737_{\pm 0.092}$
		LLM-Tree	$0.718_{\pm 0.115}$	$0.739_{\pm 0.118}$	$0.754_{\pm 0.121}$	$0.758_{\pm 0.123}$	$0.759_{\pm 0.122}$
	Same tree param. space	GradTree	$0.669_{\pm 0.105}$	$0.746_{\pm 0.118}$	$0.786_{\pm 0.117}$	$0.800_{\pm 0.115}$	$0.812_{\pm 0.117}$
	Same tree param. space	$\text{GradTree}_{\leftarrow \text{TabPFN}}$	$0.732_{\pm 0.098}$	$0.774_{\pm 0.104}$	$0.803_{\pm 0.103}$	$0.816_{\pm 0.111}$	$0.818_{\pm 0.111}$
	Same tree param. space	MotherTree	$0.733_{\pm 0.097}$	$0.764_{\pm 0.103}$	$0.787_{\pm 0.103}$	$0.795_{\pm 0.103}$	$0.795_{\pm 0.103}$
	Same tree param. space	MotherTree-Tune	$\mathbf{0.776}_{\pm 0.111}$	$\mathbf{0.804}_{\pm 0.117}$	$0.831_{\pm 0.108}$	$0.838_{\pm 0.111}$	$0.840_{\pm 0.112}$

A.6 Running Speed

Table 8 compares the average runtime (tree fitting and test set inference) of MotherTree and MotherTree-Tune with the TabPFN-distilled CART baseline on OpenML-CC18-Small. Direct MotherTree inference is consistently the fastest method on both GPU and CPU, requiring only about

Table 8: Average runtime (tree fitting and test-set inference) comparison on GPU and CPU for OpenML-CC18-Small over varying training set size.

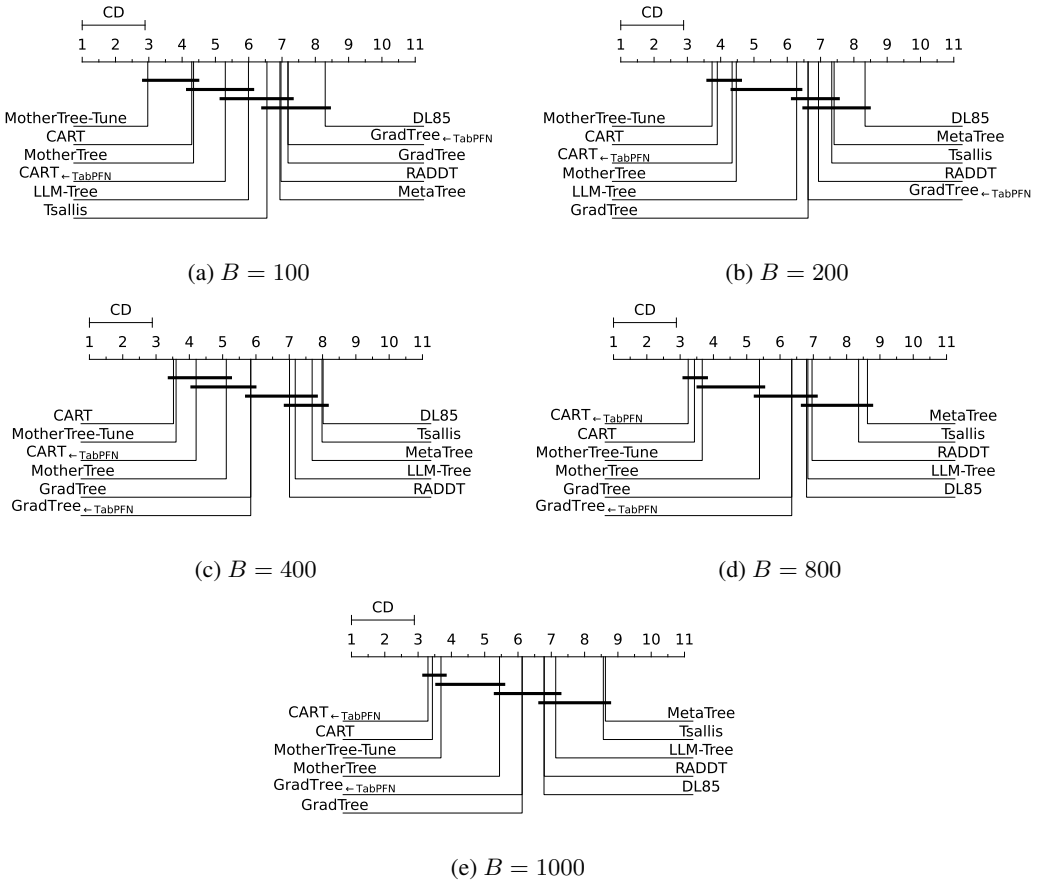
Hardware	Model	Training set budget, B 100	200	400	800	1000
GPU	$\text{CART}_{\leftarrow \text{TabPFN}}$	0.72	0.66	0.82	0.96	1.02
	MotherTree	0.15	0.15	0.18	0.19	0.20
	MotherTree-Tune	1.57	2.15	3.15	3.82	4.75
CPU	$\text{CART}_{\leftarrow \text{TabPFN}}$	1.07	1.64	2.85	4.33	4.90
	MotherTree	0.21	0.17	0.24	0.33	0.35
	MotherTree-Tune	2.25	2.85	4.03	4.96	5.47

0.15–0.20 seconds on GPU and 0.17–0.35 seconds on CPU across the evaluated training-set sizes. Its runtime also grows only mildly with the number of training samples, reflecting the fact that a task-specific tree is generated in a single forward pass.

MotherTree-Tune incurs a substantially higher cost because it performs additional task-specific optimization after tree generation. Its runtime increases with the training-set size, reaching 4.75 seconds on GPU and 5.47 seconds on CPU at $B = 1000$. This additional cost trades computational efficiency for improved predictive performance, while remaining within a few seconds per task in the evaluated settings.

A.7 Complete Critical Difference Diagram

Figure 7 contains critical difference diagrams for all models at all training set budgets, B.



(a) $B = 100$ (b) $B = 200$

(c) $B = 400$ (d) $B = 800$

(e) $B = 1000$

Figure 7: Critical difference diagrams with different maximum context lengths for two joint benchmarks

A.8 XOR BENCHMARK

A.8.1 SYNTHETIC XOR DATA SETS WITH DISTRACTOR FEATURES

Algorithm 2 XOR Data set Generation Process

Require: Number of samples N, parity bits k, number of distractors m, agreement probability q, random seed s
Ensure: Data set (X, y) and feature names $\mathcal{F}$
1: Initialize random number generator with seed s
2: Sample $B \in \{0,1\}^{N \times k}$ with $B_{ij} \sim \text{Bernoulli}(0.5)$
3: Compute parity labels $y_i \leftarrow \left(\sum_{j=1}^{k} B_{ij}\right) \bmod 2$ for $i = 1, \ldots, N$
4: $X \leftarrow B$
5: $\mathcal{F} \leftarrow \{\texttt{bit_0}, \ldots, \texttt{bit_}\{k-1\}\}$
6: **for** $d = 0$ to $m - 1$ **do**
7: Sample noise vector $\epsilon^{(d)} \in \{0,1\}^N$, where $\epsilon_i^{(d)} \sim \text{Bernoulli}(1-q)$
8: Construct distractor $z^{(d)} \leftarrow \left(y + \epsilon^{(d)}\right) \bmod 2$
9: $X \leftarrow [X \mid z^{(d)}]$
10: Append $\texttt{distract_}d$ to $\mathcal{F}$
11: **end for**
12: **return** $(X, y, \mathcal{F})$

We set $k \in \{2, 3, 4, 5\}$ parity bits, which defines the difficulty of addressing the parity problems. Number of distractors m and λ define the quantity of sub-optimal features that trees are allowed to select. $q \in \{0.50, 0.55, 0.60, 0.65, 0.70\}$ to be equals to y and is flipped otherwise. Probability q defines the correlation between distractors and desirable features. High q will mislead the model to achieve high results even if distractors themselves are not optimal. From the above-mentioned procedure, it will yield $d = k + m \in \{4, 5, 6, 7\}$ binary features and we generate $n = 2000$ samples per data set. The detailed data set generation process is illustrated in Algorithm 2 and corresponding configuration is listed in Table 9. Appendix A.8.2 provides the closed form value for AUC ceiling using only distractors.

In the experiments, we repeat running each (k, m) pair for 10 times with different seeds. We select DL85 as the optimal tree solver to gauge the gap with other tree models. GradTree and CART together with their distilled versions are used as baselines to compare with both MotherTree and MotherTree-Tune.

Table 9: XOR/parity data set configurations. The tree depth is fixed at $D = 5$ throughout. k is the number of parity bits (y = XOR of the k bits), m the number of distractor features (independent noisy copies of y, each agreeing with it with probability q), $d = k + m$ the feature dimensionality (all features binary) and $\lambda = m - (D - k)$ is the number of distractors a model must leave *unused* for the parity tree to still fit below them. $\lambda \leq 0$ indicates model can incorporate all bits even if all distractors are captured. Every (k, m) pair is generated for $q \in \{0.50, 0.55, 0.60, 0.65, 0.70\}$; the entries of the m, λ and d columns correspond positionally. One seed per configuration gives 59 data sets in total.

k (bits)	m (distractors)	λ	q	d	#data sets
2	2, 3, 4	$-1,\ 0,\ +1$	0.50–0.70	4, 5, 6	15
3	1, 2, 3	$-1,\ 0,\ +1$		4, 5, 6	15
4	1, 2, 3	$0,\ +1,\ +2$		5, 6, 7	15
5	1, 2	$+1,\ +2$		6, 7	10
2, 3, 4, 5	0 (parity control)	$-3,\ -2,\ -1,\ 0$	0.55	k	4
				total	**59**

A.8.2 DISTRACTOR-ONLY BAYES AUC CEILING

We derive the best population AUC achievable using only the distractor features. This provides a distractor-only ceiling for the synthetic parity experiments.

Let $Y \in \{0,1\}$ be balanced and let $Z = (Z_1, \ldots, Z_m) \in \{0,1\}^m$ denote the distractors. Each distractor is an independent noisy copy of the label:

$$\Pr(Z_j = Y \mid Y) = q, \qquad q > \frac{1}{2},$$

and the Z_j are conditionally independent given Y.

Define the distractor-count score

$$S(Z) = \sum_{j=1}^{m} Z_j.$$

Let

$$Z^+ \sim P(Z \mid Y = 1), \qquad Z^- \sim P(Z \mid Y = 0)$$

denote the distractor vectors of an independently drawn positive and negative example, respectively. We further define

$$S_+ := S(Z^+), \qquad S_- := S(Z^-).$$

Thus, S_+ is the number of distractors equal to one for a positive example, while S_- is the corresponding number for a negative example.

Proposition 1. *The Bayes-optimal distractor-only score $r(z) = \Pr(Y = 1 \mid Z = z)$ is a strictly increasing function of $S(z)$. Therefore, $\mathrm{AUC}^*_{\mathrm{dist}}(m, q) = \Pr(S_+ > S_-) + \frac{1}{2}\Pr(S_+ = S_-)$, where $S_+ \sim \mathrm{Binomial}(m, q), S_- \sim \mathrm{Binomial}(m, 1-q)$.*

Proof. Consider a realization $z = (z_1, \ldots, z_m)$ containing $s = \sum_j z_j$ ones. Conditional independence gives

$$\Pr(Z = z \mid Y = 1) = q^s(1-q)^{m-s},$$

and

$$\Pr(Z = z \mid Y = 0) = (1-q)^s q^{m-s}.$$

Since the classes are balanced, Bayes' rule yields

$$r(z) = \Pr(Y = 1 \mid Z = z) = \frac{q^s(1-q)^{m-s}}{q^s(1-q)^{m-s} + (1-q)^s q^{m-s}}.$$

Hence $r(z)$ depends on z only through $s = S(z)$, i.e.,

$$\Pr(Y = 1 \mid Z = z) = \Pr(Y = 1 \mid S = s).$$

Equivalently,

$$r(s) = \frac{1}{1 + \left(\frac{1-q}{q}\right)^{2s-m}}.$$

Because $q > 1/2$, $r(s)$ is strictly increasing in s.

AUC is the probability that a randomly drawn positive example receives a larger score than a randomly drawn negative example, with ties receiving half credit:

$$\mathrm{AUC} = \Pr(r(Z^+) > r(Z^-)) + \frac{1}{2}\Pr(r(Z^+) = r(Z^-)),$$

where $Z^+ \sim P(Z \mid Y = 1)$ and $Z^- \sim P(Z \mid Y = 0)$. Since r is strictly increasing in S, this is equivalent to

$$\mathrm{AUC} = \Pr(S_+ > S_-) + \frac{1}{2}\Pr(S_+ = S_-).$$

Finally, for a positive example each distractor equals 1 with probability q, whereas for a negative example it equals 1 with probability $1 - q$.

Therefore,

$$S_+ \sim \mathrm{Binomial}(m, q), \qquad S_- \sim \mathrm{Binomial}(m, 1-q),$$

which proves the result. □

For the values of m used in our experiments, the resulting closed forms are

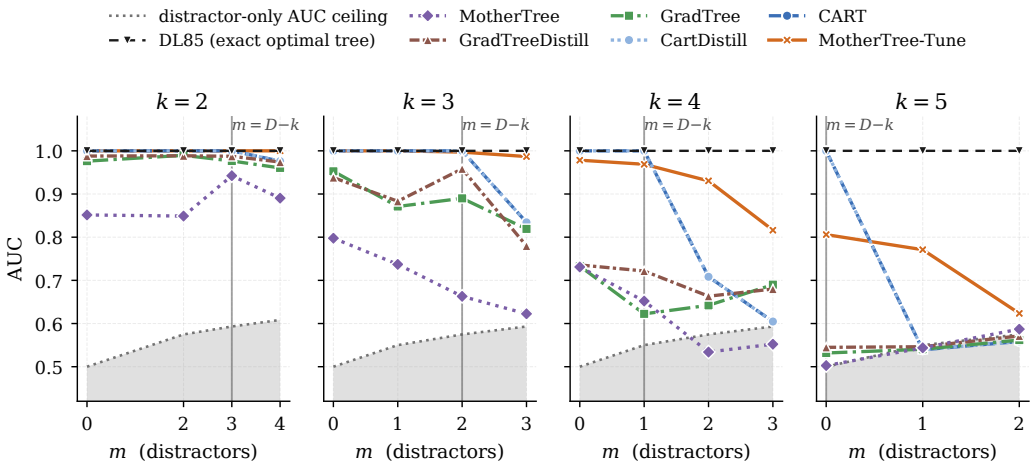
$$\mathrm{AUC}^*_{\mathrm{dist}}(1,q) = q, \qquad \mathrm{AUC}^*_{\mathrm{dist}}(2,q) = 3q^2 - 2q^3,$$
$$\mathrm{AUC}^*_{\mathrm{dist}}(3,q) = 10q^3 - 15q^4 + 6q^5, \qquad \mathrm{AUC}^*_{\mathrm{dist}}(4,q) = 35q^4 - 84q^5 + 70q^6 - 20q^7.$$

Since all k parity bits deterministically determine the label,

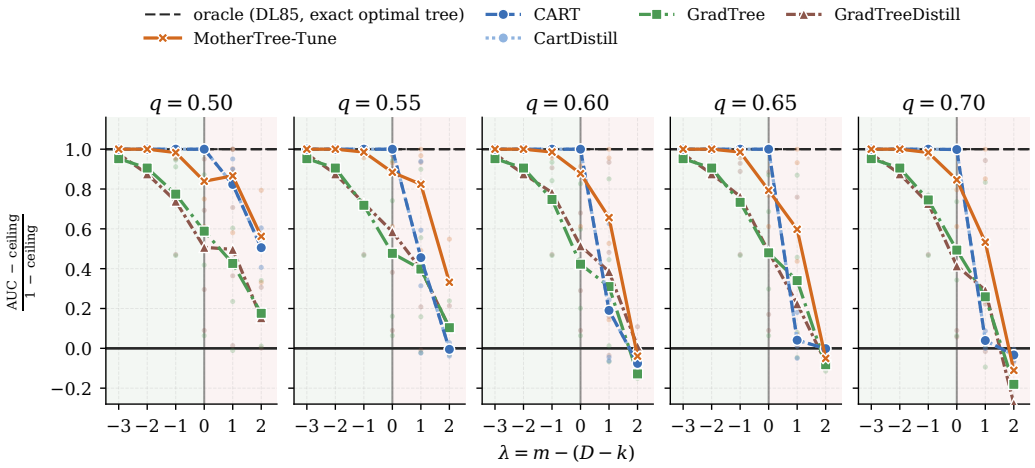
$$Y = X_1 \oplus \cdots \oplus X_k,$$

the full-feature optimum is $\mathrm{AUC}^*_{\mathrm{full}} = 1$ whenever the model class can represent the depth-k parity tree. Therefore, population AUC significantly above $\mathrm{AUC}^*_{\mathrm{dist}}(m,q)$ cannot be explained by the distractors alone and provides evidence that the predictor exploits information contained in the parity features.

A.8.3 Figures in all settings for Parity(XOR) tasks



(a) AUC against the number of distractor features m.



(b) Share of AUC under different q(label correlations)

.

Figure 8: Parity with distractors at a fixed depth budget $D = 5$. $\lambda = m - (D - k)$ is the number of distractors that must be left to learn the parity tree. Green ($\lambda \leq 0$): the depth budget has slack. Splitting on a distractor does not destroy the label. Red ($\lambda > 0$): the budget is exhausted, so each distractor used takes a level away from the parity tree; at least λ of them must be refused.