GradTree: Learning Axis-Aligned Decision Trees with Gradient Descent

Sascha Marton
University of Mannheim
68131 Mannheim
`sascha.marton@uni-mannheim.de`

Stefan Lüdtke
University of Rostock
18051 Rostock
`stefan.luedtke@uni-rostock.de`

Christian Bartelt
University of Mannheim
68131 Mannheim
`christian.bartelt@uni-mannheim.de`

Heiner Stuckenschmidt
University of Mannheim
68131 Mannheim
`heiner.stuckenschmidt@uni-mannheim.de`

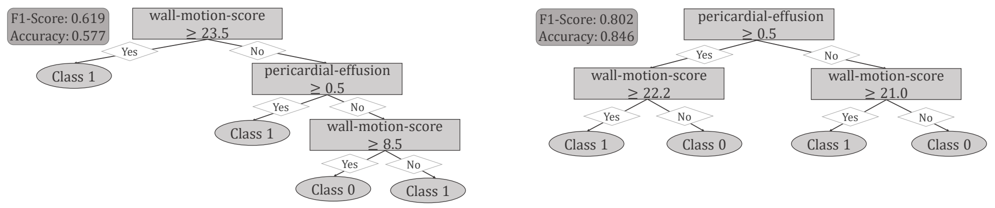
Abstract

Decision Trees (DTs) are commonly used for many machine learning tasks due to their high degree of interpretability. However, learning a DT from data is a difficult optimization problem, as it is non-convex and non-differentiable. Therefore, common approaches learn DTs using a greedy growth algorithm that minimizes the impurity locally at each internal node. Unfortunately, this greedy procedure can lead to inaccurate trees. In this paper, we present a novel approach for learning hard, axis-aligned DTs with gradient descent. The proposed method uses backpropagation with a straight-through operator on a dense DT representation, to jointly optimize all tree parameters. Our approach outperforms existing methods on a wide range of binary classification benchmarks and is available under: `https://github.com/s-marton/GradTree`

1 Introduction

Decision trees (DTs) are some of the most popular machine learning models and are still frequently used today. In particular, with the growing interest in explainable artificial intelligence (XAI), DTs have regained popularity due to their interpretability. However, learning a DT is a difficult optimization problem, since it is non-convex and non-differentiable. Therefore, the prevailing approach to learn a DT is a greedy procedure that minimizes the impurity at each internal node. The algorithms still in use today, such as CART [7] and C4.5 [34], were developed in the 1980s and have remained largely unchanged since then. Unfortunately, a greedy algorithm optimizes the objective locally at each internal node which constrains the search space, and potentially leads to inaccurate trees. We illustrate this issue below:

Example 1 The Echocardiogram dataset [11] deals with predicting one-year survival of patients after a heart attack based on tabular data from an echocardiogram. Figure 1 shows two DTs. The tree on the left is learned by a greedy algorithm (CART) while the one on the right is learned with our gradient-based approach. We can observe that the greedy procedure leads to a tree with a significantly lower performance. Splitting on the *wall-motion-score* is the locally optimal split (see Figure 1a), but globally, it is beneficial to split based on the *wall-motion-score* with different values conditioned on the *pericardial-effusion* in the second level (Figure 1b).



(a) Greedy DT (CART) (b) Gradient-Based DT (GradTree)

Figure 1: **Greedy vs. Gradient-Based DT.** Two DTs trained on the Echocardiogram dataset. The CART DT (left) makes only locally optimal splits, while GradTree (right) jointly optimizes all parameters, leading to significantly better performance.

In this paper, we propose a novel approach for learning hard, axis-aligned DTs based on a joint optimization of all tree parameters using gradient descent, which we call **Grad**ient-Based Decision **Tree** (GradTree). Similar to optimization in neural networks, GradTree yields a desirable local optimum of parameters that generalizes well to test data. Using a gradient-based optimization, GradTree can overcome the limitations of greedy approaches, which are constrained by sequentially selecting optimal splits, as illustrated in Figure 1. At the same time, GradTree can converge to a local optimum that offers good generalization, and thus provides an advantage over alternative non-greedy methods like optimal DTs [10, 1], which often suffer from severe overfitting [39]. Specifically, our contributions are:

- We introduce a dense DT representation that enables a joint, gradient-based optimization of all tree parameters (Section 3.2).
- We present a procedure to deal with the non-differentiable nature of DTs using backpropagation with a straight-through (ST) operator (Section 3.3).
- We propose a novel tree routing that allows an efficient, parallel optimization of all tree parameters with gradient descent (Section 3.4).

We empirically evaluate GradTree on a large number of real-world binary classification datasets (Section 4) and show that GradTree outperforms existing methods. On several benchmark datasets, the performance difference between GradTree and other methods is substantial. The gradient-based optimization also provides more flexibility by allowing split adjustments during training and easy integration of custom loss functions.

2 Related Work

Greedy DT Algorithms The most prominent DT learning algorithms still frequently used, namely CART [7] and C4.5 [34], date back to the 1980s. Both follow a greedy procedure to learn a DT. Since then, many variations to those algorithms have been proposed, for instance C5.0 [21] and GUIDE [25, 26]. However, until today, none of these algorithms was able to consistently outperform CART and C4.5 as shown for instance by Zharmagambetov et al. [40].

Optimal DTs To overcome the issues of a greedy DT induction, many researchers focused on finding an efficient alternative. Optimal DTs aim to optimize an objective (e.g., the purity) through an approximate brute force search to find a globally optimal tree with a certain specification [40]. Therefore, they most commonly use mixed integer optimization [5] or a branch-and-bound algorithm to remove irrelevant parts from the search space [1, 23]. MurTree [10] further uses dynamic programming, which reduces the runtime significantly. However, most state-of-the-art approaches still require binary data and therefore a discretization of continuous features [5, 1, 10], which can lead to information loss. An exception is the approach by Mazumder et al. [27], which can handle continuous features out-of-the-box. However, their method is optimized for very sparse trees and limited to a maximum depth of 3.

While optimal DTs search for a global optimum, GradTree does not necessarily pursue this. Instead, like optimization in neural networks, it aims for a local optimum that offers good generalization to test data. We further want to emphasize that the local optima that can be reached by GradTree have a significant advantage over the local optimum of a greedy approach: While the local optimum of greedy approaches is constrained by sequentially selecting the optimal split at each node, GradTree overcomes this limitation by optimizing all parameters jointly.

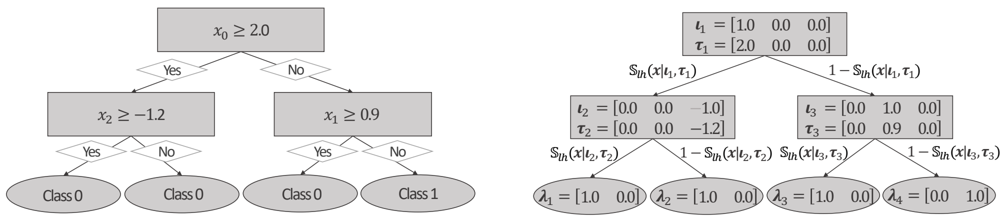
Genetic DTs Another way to learn DTs in a non-greedy fashion is by using evolutionary algorithms. Evolutionary algorithms perform a robust global search in the space of candidate solutions based on the concept of survival of the fittest [3]. This usually results in smaller trees and a better identification of feature interactions compared to a greedy, local search [12].

Oblique DTs In contrast to vanilla DTs that make a hard decision at each internal node, many hierarchical mixture of expert models [17] have been proposed. They usually make soft splits, where each branch is associated with a probability [14, 13]. Further, the models do not comprise univariate, axis-aligned splits, but are oblique with respect to the axes. These adjustments to the tree architecture allow for the application of further optimization algorithms, including gradient descent. Blanquero et al. [6] aim to increase the interpretability of oblique trees by optimizing for sparsity, using fewer variables at each split and simultaneously fewer splits in the whole tree. Tanno et al. [35] combine the benefits of neural networks and DTs, using so-called adaptive neural trees (ANTs). They employ a stochastic routing based on a Bernoulli distribution and utilize non-linear transformer modules at the edges, making the resulting trees soft and oblique. Xu et al. [36] propose One-Stage Tree as a novel method for learning soft DTs, including the tree structure, while maintaining discretization during training, which results in a higher interpretability compared to existing soft DTs. However, in contrast to GradTree, the routing is instance-wise, which significantly hampers a global interpretation of the model. Norouzi et al. [29] proposed an approach to overcome the need for soft decisions to apply gradient-based algorithms by minimizing a convex-concave upper bound on the tree's empirical loss. While this allows the use of hard splits, the approach is still limited to oblique trees. Zantedeschi et al. [39] use argmin differentiation to simultaneously learn all tree parameters by relaxing a mixed-integer program for discrete parameters to allow for gradient-based optimization. This allows hard splits, but in contrast to GradTree, they still require a differentiable split function (e.g., a linear function which results in oblique trees). Similarly, Karthikeyan et al. [18] developed a gradient-based approach to learn hard DTs. Like to GradTree, they use an ST operator to handle the hard step functions. While their formulation is limited to oblique trees, our approach permits axis-aligned DTs.

In summary, unlike oblique DTs, GradTree allows hard, axis-aligned splits that consider only a single feature at each split, providing significantly higher interpretability, especially at the split-level. This is supported by Molnar [28] where the authors argue that humans cannot comprehend explanations involving more than three dimensions at once.

Oblivious DT Ensembles Popov et al. [32] proposed an oblivious tree ensemble for deep learning. Oblivious DTs use the same splitting feature and threshold in all internal nodes of the same depth, making them only suitable as weak learners in an ensemble. They use an entmax transformation of the choice function and a two-class entmax as split function. This results in oblique and soft trees, while GradTree is axis-aligned and hard. Chang et al. [8] proposed a temperature annealing procedure to gradually turn the input to an entmax function one-hot, which can enforce axis-aligned trees. In contrast, our approach employs an ST operator immediately following an entmax transformation to yield a one-hot encoded vector. Our experiments substantiate that our method achieves superior results in the context of individual DTs.

Deep Neural Decision Trees (DNDTs) Yang et al. [37] propose DNDTs that realize tree models as neural networks, utilizing a soft binning function for splitting. Therefore, the resulting trees are soft, but axis-aligned, which makes this work closely related to our approach. Since DNDTs are generated via the Kronecker product of the binning layers, the structure depends on the number of features and classes (and the number of bins). As discussed by the authors, this results in poor scalability w.r.t. the number of features, which currently can only be solved by using random forests for high-dimensional datasets (> 12 features). Our approach, in contrast, scales linearly with the number of features, making it efficient for high-dimensional datasets. Furthermore, using the Kronecker product to build the tree prevents splitting on the same feature with different thresholds in the same path, which can be crucial to achieve a good performance. For GradTree, both the split threshold and the split index are learned parameters, inherently allowing the model to split on the same feature multiple times.



$x_0 \geq 2.0$
Yes
No
$x_2 \geq -1.2$
$x_1 \geq 0.9$
Yes
No
Yes
No
Class 0
Class 0
Class 0
Class 1
$\iota_1 = [1.0 \quad 0.0 \quad 0.0]$
$\tau_1 = [2.0 \quad 0.0 \quad 0.0]$
$\mathbb{S}_{lh}(x|\iota_1, \tau_1)$
$1 - \mathbb{S}_{lh}(x|\iota_1, \tau_1)$
$\iota_2 = [0.0 \quad 0.0 \quad 1.0]$
$\tau_2 = [0.0 \quad 0.0 \quad -1.2]$
$\iota_3 = [0.0 \quad 1.0 \quad 0.0]$
$\tau_3 = [0.0 \quad 0.9 \quad 0.0]$
$\mathbb{S}_{lh}(x|\iota_2, \tau_2)$
$1 - \mathbb{S}_{lh}(x|\iota_2, \tau_2)$
$\mathbb{S}_{lh}(x|\iota_3, \tau_3)$
$1 - \mathbb{S}_{lh}(x|\iota_3, \tau_3)$
$\lambda_1 = [1.0 \quad 0.0]$
$\lambda_2 = [1.0 \quad 0.0]$
$\lambda_3 = [1.0 \quad 0.0]$
$\lambda_4 = [0.0 \quad 1.0]$

(a) Vanilla DT Representation (b) Dense DT Representation

Figure 2: **Standard vs. Dense DT Representation.** Comparison of a standard and the equivalent dense representation for an exemplary DT with depth 2 and a dataset with 3 variables and 2 classes. Here, $\mathbb{S}_{\text{lh}}$ stands for $\mathbb{S}_{\text{logistic\_hard}}$ (Equation 7).

3 GradTree: Gradient-Based Decision Trees

In this section, we present a new DT representation and a novel algorithm that allows learning hard, axis-aligned DTs with gradient descent. More specifically, we use backpropagation with a straight-through (ST) operator (Section 3.3) on a dense DT representation (Section 3.2) to adjust the model parameters during the training. Furthermore, our novel tree routing (Section 3.4) allows an efficient optimization of all parameters over an entire batch with a single set of matrix operations.

3.1 Arithmetic Decision Tree Formulation

Here, we introduce a notation for DTs with respect to their parameters. We formulate DTs as an arithmetic function based on addition and multiplication, rather than as a nested concatenation of rules, which is necessary for a gradient-based learning. Note that our notation and training procedure assume fully-grown (i.e. complete, full) DTs. After training, we apply a basic post-hoc pruning to reduce the tree size for application. Our formulation aligns with Kontschieder et al. [20]. However, they only consider stochastic routing and oblique trees, whereas our formulation emphasizes deterministic routing and axis-aligned trees.

For a DT of depth d, the parameters include one split threshold and one feature index for each internal node, represented as vectors $\boldsymbol{\tau} \in \mathbb{R}^{2^d-1}$ and $\boldsymbol{\iota} \in \mathbb{N}^{2^d-1}$ respectively, where $2^d - 1$ equals the number of internal nodes. Additionally, each leaf node comprises a class membership, in the case of a classification task, which we denote as the vector $\boldsymbol{\lambda} \in \mathcal{C}^{2^d}$, where $\mathcal{C}$ is the set of classes and 2^d equals the number of leaf nodes.

Formally, a DT can be expressed as a function $DT(\cdot|\boldsymbol{\tau}, \boldsymbol{\iota}, \boldsymbol{\lambda}) : \mathbb{R}^n \to \mathcal{C}$ with respect to its parameters:

$$DT(\boldsymbol{x}|\boldsymbol{\tau}, \boldsymbol{\iota}, \boldsymbol{\lambda}) = \sum_{l=0}^{2^d-1} \lambda_l \, \mathbb{L}(\boldsymbol{x}|l, \boldsymbol{\tau}, \boldsymbol{\iota}) \tag{1}$$

The function $\mathbb{L}(\boldsymbol{x}|l, \boldsymbol{\tau}, \boldsymbol{\iota}) : \mathbb{R}^n \to \{0, 1\}$ indicates whether a sample $\boldsymbol{x} \in \mathbb{R}^n$ belongs to a leaf l, and can be defined as a multiplication of the split functions of the preceding internal nodes. We define the split function $\mathbb{S}$ as a Heaviside step function

$$\mathbb{S}_{\text{Heaviside}}(\boldsymbol{x}|\iota, \tau) = \begin{cases} 1, \text{if } x_\iota \geq \tau \\ 0, \text{otherwise} \end{cases} \tag{2}$$

where ι is the index of the feature considered at a certain split and τ is the corresponding threshold. By enumerating the internal nodes of a fully-grown tree with depth d in a breadth-first order, we can now define the indicator function $\mathbb{L}$ for a leaf l as

$$\mathbb{L}(\boldsymbol{x}|l, \boldsymbol{\tau}, \boldsymbol{\iota}) = \prod_{j=1}^{d} \left(1 - \mathfrak{p}(l,j)\right) \, \mathbb{S}(\boldsymbol{x}|\tau_{\mathfrak{i}(l,j)}, \iota_{\mathfrak{i}(l,j)}) + \mathfrak{p}(l,j) \, \left(1 - \mathbb{S}(\boldsymbol{x}|\tau_{\mathfrak{i}(l,j)}, \iota_{\mathfrak{i}(l,j)})\right) \tag{3}$$

Here, $\mathfrak{i}$ is the index of the internal node preceding a leaf node l at a certain depth j and can be calculated as

$$\mathfrak{i}(l,j) = 2^{j-1} + \left\lfloor \frac{l}{2^{d-(j-1)}} \right\rfloor - 1 \tag{4}$$

Additionally, $\mathfrak{p}$ indicates whether the left ($\mathfrak{p} = 0$) or the right branch ($\mathfrak{p} = 1$) was taken at the internal node preceding a leaf node l at a certain depth j. We can calculate $\mathfrak{p}$ as

$$\mathfrak{p}(l, j) = \left\lfloor \frac{l}{2^{d-j}} \right\rfloor \bmod 2 \tag{5}$$

As becomes evident, DTs involve non-differentiable operations in terms of the split function, including the split feature selection (Equation 2), which precludes the application of backpropagation. Specifically, to efficiently learn a DT using backpropagation, we must address three challenges:

C1 The index ι for the split feature selection is defined as $\iota \in \mathbb{N}$. However, the index ι is a parameter of the DT and a gradient-based optimization requires $\iota \in \mathbb{R}$.

C2 The split function $\mathbb{S}(\boldsymbol{x}|\iota, \tau)$ is a Heaviside step function with an undefined gradient for $x_\iota = \tau$ and 0 gradient elsewhere, which precludes an efficient optimization.

C3 Leafs in a vanilla DT comprise a class membership $\lambda \in \mathcal{C}$. To calculate an informative loss and optimize the leaf parameters with gradient descent, we need $\lambda \in \mathbb{R}^c$ where c is the number of classes.

Additionally, the computation of the internal node index $\mathfrak{i}$ and path position $\mathfrak{p}$ involves non-differentiable operations. However, given our focus on fully-grown trees, these values remain constant, allowing for their computation prior to the optimization process.

3.2 Dense Decision Tree Representation

In this subsection, we present a differentiable representation of the feature indices $\boldsymbol{\iota}$ to facilitate gradient-based optimization, which is illustrated in Figure 2.

To this end, we expand the vector $\boldsymbol{\iota} \in \mathbb{R}^{2^d-1}$ to a matrix $I \in \mathbb{R}^{2^d-1} \times \mathbb{R}^n$. This is achieved by one-hot encoding the feature index as $\boldsymbol{\iota} \in \mathbb{R}^n$ for each internal node. This adjustment is necessary for the optimization process to account for the fact that feature indices are categorical instead of ordinal. Although our matrix representation for feature selection has parallels with that proposed by Popov et al. [32], we introduce a novel aspect: A matrix representation for split thresholds. We denote this representation as $T \in \mathbb{R}^{2^d-1} \times \mathbb{R}^n$. Instead of representing a single value for all features, we store individual values for each feature, denoted as $\boldsymbol{\tau} \in \mathbb{R}^n$. This modification is tailored to support the optimization process, particularly in recognizing that split thresholds are feature-specific and non-interchangeable. In essence, a viable split threshold for one feature may not be suitable for another. This adjustment acts as a memory mechanism, ensuring that a given split threshold is exclusively associated with the corresponding feature. Consequently, this refinement enhances the exploration of feature selection at every split during the training. We can now reformulate the Heaviside step function (Equation 2) as

$$\mathbb{S}_{\text{logistic}}(\boldsymbol{x}|\boldsymbol{\iota}, \boldsymbol{\tau}) = S\left(\sum_{i=0}^{n} \iota_i x_i - \sum_{i=0}^{n} \iota_i \tau_i\right) = S\left(\boldsymbol{\iota} \cdot \boldsymbol{x} - \boldsymbol{\iota} \cdot \boldsymbol{\tau}\right) \tag{6}$$

$$\mathbb{S}_{\text{logistic\_hard}}(\boldsymbol{x}|\boldsymbol{\iota}, \boldsymbol{\tau}) = \lfloor \mathbb{S}_{\text{logistic}}(\boldsymbol{x}|\boldsymbol{\iota}, \boldsymbol{\tau}) \rceil \tag{7}$$

where $S(x) = \frac{1}{1+e^{-x}}$ denotes the logistic function and $\lfloor \cdot \rceil$ represents for rounding to the nearest integer. In our context, with $\boldsymbol{\iota}$ being one-hot encoded, $\mathbb{S}_{\text{logistic\_hard}}(\boldsymbol{x}|\boldsymbol{\iota}, \boldsymbol{\tau}) = \mathbb{S}_{\text{Heaviside}}(\boldsymbol{x}|\iota, \tau)$ holds.

3.3 Backpropagation of Decision Tree Loss

While the dense representation emphasizes an efficient learning of axis-aligned DTs, it does not solve **C1**-**C3**. In this subsection, we will address those challenges by using the ST operator.

For the function value calculation in the forward pass, we need to assure that $\boldsymbol{\iota}$ is a one-hot encoded vector. This can be achieved by applying a hardmax function on the feature index vector for each internal node. However, applying a hardmax is a non-differentiable operation, which precludes gradient computation. To overcome this issue, we use the ST operator [4]: For the forward pass, we apply the hardmax as is. For the backward pass, we exclude this operation and directly propagate back the gradients of $\boldsymbol{\iota}$. Accordingly, we can optimize the parameters of $\boldsymbol{\iota}$ where $\boldsymbol{\iota} \in \mathbb{R}$ while still using axis-aligned splits during training (**C1**). However, this procedure introduces a mismatch between the forward and backward pass. To reduce this mismatch, we additionally perform an entmax transformation [31] to generate a sparse distribution over $\boldsymbol{\iota}$ before applying the hardmax.

Similarly, we employ the ST operator to ensure hard splits (Equation 7) by excluding $\lfloor \cdot \rceil$ for the backward pass (**C2**). Using the sigmoid logistic function before applying the ST operator (see Equation 6) utilizes the distance to the split threshold as additional information for the gradient calculation. If the feature considered at an internal node is close to the split threshold for a specific sample, this will result in smaller gradients compared to a sample that is more distant.

Furthermore, we need to adjust the leaf nodes of the DT to allow an efficient loss calculation (**C3**). Vanilla DTs contain the predicted class for each leaf node and are functions $DT : \mathbb{R}^n \to \mathcal{C}$. We use logits at each leaf node and therefore define DTs as $DT : \mathbb{R}^n \to \mathbb{R}^c$ where c is the number of classes. We can then convert the logits to a probability distribution over the classes by applying a softmax transformation. Consequently, the parameters of the leaf nodes are defined as $L \in \mathbb{R}^{2^n} \times \mathbb{R}^c$ for the whole tree and $\boldsymbol{\lambda} \in \mathbb{R}^c$ for a specific leaf node. This adjustment allows the application of standard loss functions.

3.4 Deterministic Tree Routing and Training

In the previous subsections, we introduced the adjustments that are necessary to apply gradient descent to DTs. During the optimization, we calculate the gradients with backpropagation. The tree pass function to calculate the function values is summarized in Algorithm 1.Our tree routing facilitates the computation of the tree pass function over a complete batch as a single set of matrix operations, which allows an efficient computation. We also want to note that our dense representation can always be converted into an equivalent vanilla DT representation. Similarly, the fully-grown nature of GradTree is only required during the gradient-based optimization and standard pruning techniques to reduce the tree size are applied post-hoc.

Algorithm 1 Tree Pass Function

```
function PASS($I, T, L, \boldsymbol{x}$)
    $I \leftarrow \text{entmax}(I)$
    $I \leftarrow I - c_1^*$   where $c_1^* = I - \text{hardmax}(I)$ ▷ ST operator
    $\hat{\boldsymbol{y}} \leftarrow [0]^c$
    for $l = 0, \dots, 2^d - 1$ do
        $p \leftarrow 1$
        for $j = 1, \dots, d$ do
            $\mathfrak{i} \leftarrow 2^{j-1} + \left\lfloor \frac{l}{2^{d-(j-1)}} \right\rfloor - 1$   ▷ Equation 4
            $\mathfrak{p} \leftarrow \left\lfloor \frac{l}{2^{d-j}} \right\rfloor \bmod 2$   ▷ Equation 5
            $s \leftarrow S\left(\sum_{i=0}^{n} T_{\mathfrak{i},i}\, I_{\mathfrak{i},i} - \sum_{i=0}^{n} x_i\, I_{\mathfrak{i},i}\right)$   ▷ Equation 6
            $s \leftarrow s - c_2^*$   where $c_2^* = s - \lfloor s \rceil$ ▷ ST operator
            $p \leftarrow p\, ((1 - \mathfrak{p})\, s + \mathfrak{p}\, (1 - s))$   ▷ Equation 3
        end for
        $\hat{\boldsymbol{y}} \leftarrow \hat{\boldsymbol{y}} + L_l\, p$   ▷ Equation 1
    end for
    return $\sigma(\hat{\boldsymbol{y}})$ ▷ Softmax $\sigma$ to get probability distribution
end function
```

Furthermore, our implementation optimizes the gradient descent algorithm by leveraging common stochastic gradient descent techniques, including mini-batch calculation and momentum using the Adam optimizer [19] with weight averaging [15]. Moreover, we implement early stopping and random restarts to avoid bad initial parametrizations, where the best parameters are selected based on the validation loss. Further details can be found in Appendix C.

4 Experimental Evaluation

Datasets and Preprocessing The experiments were conducted on several benchmark datasets, mainly from the UCI repository [11]. For all datasets, we performed a standard preprocessing: Similar to Popov et al. [32], we applied leave-one-out encoding to all categorical features and further performed a quantile transform, making each feature follow a normal distribution. We used a $80\%/20\%$ train-test split for all datasets. To account for class imbalance, we rebalanced datasets using SMOTE [9] if the minority class accounts for less than 25% of the data. For GradTree and DNDT, we used 20% of the training data as validation data for early stopping. As DL8.5 requires binary features, we discretized numeric features using quantile binning with 5 bins and one-hot encoded categorical features. Details and sources of the datasets are available in Appendix D.

Table 1: **Binary Classification Performance.** We report macro F1-scores (mean ± stdev over 10 trials) on test data with optimized hyperparameters. The rank of each method is presented in brackets. The datasets are sorted by the number of features.

	Gradient-Based		Non-Greedy		Greedy
	GradTree (ours)	DNDT	GeneticTree	DL8.5 (Optimal)	CART
Blood Transfusion	**0.628 ± .036 (1)**	0.543 ± .051 (5)	0.575 ± .094 (4)	0.590 ± .034 (3)	0.613 ± .044 (2)
Banknote Authentication	**0.987 ± .007 (1)**	0.888 ± .013 (5)	0.922 ± .021 (4)	0.962 ± .011 (3)	0.982 ± .007 (2)
Titanic	**0.776 ± .025 (1)**	0.726 ± .049 (5)	0.730 ± .074 (4)	0.754 ± .031 (2)	0.738 ± .057 (3)
Raisins	0.840 ± .022 (4)	0.821 ± .033 (5)	**0.857 ± .021 (1)**	0.849 ± .027 (3)	0.852 ± .017 (2)
Rice	0.926 ± .007 (3)	0.919 ± .012 (5)	0.927 ± .005 (2)	0.925 ± .008 (4)	**0.927 ± .006 (1)**
Echocardiogram	**0.658 ± .113 (1)**	0.622 ± .114 (3)	0.628 ± .105 (2)	0.609 ± .112 (4)	0.555 ± .111 (5)
Wisconcin Breast Cancer	0.904 ± .022 (2)	**0.913 ± .032 (1)**	0.892 ± .028 (4)	0.896 ± .021 (3)	0.886 ± .025 (5)
Loan House	**0.714 ± .041 (1)**	0.694 ± .036 (2)	0.451 ± .086 (5)	0.607 ± .045 (4)	0.662 ± .034 (3)
Heart Failure	0.750 ± .070 (3)	0.754 ± .062 (2)	0.748 ± .068 (4)	0.692 ± .062 (5)	**0.775 ± .054 (1)**
Heart Disease	**0.779 ± .047 (1)**	$n > 12$	0.704 ± .059 (4)	0.722 ± .065 (2)	0.715 ± .062 (3)
Adult	0.743 ± .034 (2)	$n > 12$	0.464 ± .055 (4)	0.723 ± .011 (3)	**0.771 ± .011 (1)**
Bank Marketing	**0.640 ± .027 (1)**	$n > 12$	0.473 ± .002 (4)	0.502 ± .011 (3)	0.608 ± .018 (2)
Congressional Voting	**0.950 ± .021 (1)**	$n > 12$	0.942 ± .021 (2)	0.924 ± .043 (4)	0.933 ± .032 (3)
Absenteeism	**0.626 ± .047 (1)**	$n > 12$	0.432 ± .073 (4)	0.587 ± .047 (2)	0.564 ± .042 (3)
Hepatitis	0.608 ± .078 (2)	$n > 12$	0.446 ± .024 (4)	0.586 ± .083 (3)	**0.622 ± .078 (1)**
German	**0.592 ± .068 (1)**	$n > 12$	0.412 ± .006 (4)	0.556 ± .035 (3)	0.589 ± .065 (2)
Mushroom	**1.000 ± .001 (1)**	$n > 12$	0.984 ± .003 (4)	0.999 ± .001 (2)	0.999 ± .001 (3)
Credit Card	0.674 ± .014 (4)	$n > 12$	**0.685 ± .004 (1)**	0.679 ± .007 (3)	0.683 ± .010 (2)
Horse Colic	**0.842 ± .039 (1)**	$n > 12$	0.496 ± .169 (4)	0.708 ± .038 (3)	0.786 ± .062 (2)
Thyroid	0.905 ± .010 (2)	$n > 12$	0.605 ± .116 (4)	0.682 ± .018 (3)	**0.922 ± .011 (1)**
Cervical Cancer	**0.521 ± .043 (1)**	$n > 12$	0.514 ± .034 (2)	0.488 ± .027 (4)	0.506 ± .034 (3)
Spambase	0.903 ± .025 (2)	$n > 12$	0.863 ± .019 (3)	0.863 ± .011 (4)	**0.917 ± .011 (1)**
Mean Relative Diff. (MRD) ↓	**0.008 ± .012 (1)**	0.056 ± .051 (3)	0.211 ± .246 (5)	0.084 ± .090 (4)	0.035 ± .048 (2)
Mean Reciprocal Rank (MRR) ↑	**0.758 ± .306 (1)**	0.370 ± .268 (3)	0.365 ± .228 (4)	0.335 ± .090 (5)	0.556 ± .293 (2)

Methods We compared GradTree to the most prominent approach from each category (see Section 2) to ensure a concise, yet holistic evaluation focusing on hard, axis-aligned DTs. Specifically, we selected the following methods:

- **CART**: We use the sklearn [30] implementation, which uses an optimized version of the CART algorithm. CART typically employs the Gini impurity measure, but we additionally allowed entropy.
- **Evolutionary DTs**: We use GeneticTree [33] for learning of DTs with a genetic algorithm.
- **DNDT**: We use the official DNDT implementation [38]. For a fair comparison, we enforce binary trees by setting the number of cut points to 1 and ensure hard splits during inference. As suggested by Yang et al. [37], we limited DNDTs to datasets with no more than 12 features, due to scalability issues.
- **DL8.5 (Optimal DTs)**: We use the official DL8.5 implementation [2] including improvements from MurTree [10] which reduces the runtime significantly.

GradTree is implemented in Python using TensorFlow[1]. To ensure a fair comparison, we further applied a simple post-hoc pruning for GradTree to remove all branches with zero samples based on one pass of the training data. Similar to DNDT, we used a cross-entropy loss.

Hyperparameters We conducted a random search with cross-validation to determine the optimal hyperparameters. The complete list of relevant hyperparameters for each approach along with additional details on the selection are in Appendix C.

4.1 Results

GradTree outperforms existing DT learners First, we evaluated the performance of GradTree against existing methods on the benchmark datasets in terms of the macro F1-Score, which inherently considers class imbalance. We report the relative difference to the best model (MRD) and mean reciprocal rank (MRR), following the approach of Yang et al. [37]. Overall, GradTree outperformed existing approaches for binary classification tasks (best MRD of 0.008 and MRR of 0.758). More specifically, GradTree significantly outperformed state-of-the-art non-greedy DT methods, including

[1]The code of our implementation is available under: `https://github.com/s-marton/GradTree`.

Table 2: **Summarized Results.** Top: Average tree size. Mid: Mean difference between train and test performance as overfitting indicator. Bottom: Mean reciprocal rank with default parameters for performance comparison. Detailed results are in Appendix B.

	GradTree	DNDT	GeneticTree	DL8.5	CART
Tree Size	54	887	7	28	67
Train-Test Difference	0.051	0.039	0.204	0.202	0.183
Default Setting (MRR ↑)	**0.670**	0.306	0.427	0.371	0.571

DNDTs as our gradient-based benchmark. Further, GradTree demonstrated superior performance over CART, achieving the best performance on 13 datasets as compared to only 6 for CART. Notably, the performance difference between GradTree and existing methods was substantial for several datasets, such as *Echocardiogram*, *Heart Disease* and *Absenteeism*.

GradTree has a small effective tree size The effective tree size (= size after pruning) of GradTree is smaller than CART (Table 2). Only the tree size for GeneticTree is significantly smaller, which is caused by the complexity penalty of the genetic algorithm. DL8.5 also has a smaller average tree size than CART and GradTree. We can attribute this to DL8.5 being only feasible up to a depth of 4 due to the high computational complexity. The tree size for DNDTs scales with the number of features (and classes) which quickly results in large trees. Furthermore, pruning DNDTs is non-trivial due to the use of the Kronecker product (it is not sufficient to prune subtrees bottom-up).

GradTree is robust to overfitting We can observe that gradient-based approaches were more robust and less prone to overfitting compared to a greedy optimization with CART and alternative non-greedy methods. We measure overfitting by the difference between the mean train and test performance (see Table 2). GradTree exhibits a train-test performance difference of 0.051, considerably smaller than that of CART (0.183), GeneticTree (0.204), and DL8.5 (0.202). DNDTs, which are also gradient-based, achieved an even smaller difference of 0.039.

GradTree does not rely on extensive hyperparameter optimization Besides their interpretability, a distinct advantage of DTs is that they typically do not rely on an extensive hyperparameter optimization. We show that the same is true for GradTree by evaluating the performance with default configurations (see Table 2). When using the default parameters, GradTree still outperformed existing methods (highest MRR and most wins).

GradTree is efficient for large and high-dimensional datasets For each dataset, a greedy optimization using CART was substantially faster than other methods, taking less than a second. Nevertheless, for most datasets, training GradTree took less than 30 seconds (mean runtime of 45 seconds). DNDT had comparable runtimes to GradTree. For most datasets, DL8.5 had a low runtime of less than 10 seconds. However, scalability issues become apparent with DL8.5, especially with an increasing number of features and samples. Its runtime notably surpassed GradTree on various datasets, taking for instance around 300 seconds for *Credit Card*. The complete runtimes for each dataset are listed in the appendix (Table 8).

ST entmax outperforms alternative methods In an ablation study (see Table 3 for a summary), we evaluated our method of utilizing an ST operator directly after an entmax transformation to address the non-differentiability of DTs. We contrasted this against alternative strategies used in related work. Our approach notably surpassed ST Gumbel Softmax [16] and outperformed the temperature annealing technique proposed by Chang et al. [8] to gradually turn the entmax one-hot.

Figure 3: **Ablation Study Summary.** We compare our approach to deal with the non-differentiable nature of DTs with alternative methods, reporting the average macro F1-scores over 10 trials. The complete results are listed in Appendix B.

	ST Entmax (ours)	ST Gumbel	Temp. Annealing
Default	**0.764**	0.560	0.757
Optimized	**0.771**	0.569	0.759

5 Conclusion and Future Work

In this paper, we proposed GradTree, a novel method for learning hard, axis-aligned DTs based on a joint optimization of all tree parameters with gradient descent. Our empirical evaluations indicate that

GradTree excels over existing methods in binary tasks. The substantial performance increase achieved by GradTree across multiple datasets highlights its importance as a noteworthy contribution to the existing repertoire of DT learning methods. Moreover, gradient-based optimization provides greater flexibility, allowing for easy integration of custom loss functions tailored to specific application scenarios. Another advantage is the ability to relearn the threshold value as well as the split index. Therefore, GradTree is suitable for dynamic environments, such as online learning tasks.

Currently, GradTree employs conventional post-hoc pruning. In future work, we want to consider pruning already during the training, for instance through a learnable choice parameter to decide if a node is pruned, similar to Zantedeschi et al. [39]. Although our focus was on stand-alone DTs aiming for intrinsic interpretability, GradTree holds potential as a foundational method for learning hard, axis-aligned tree ensembles end-to-end via gradient descent. Exploring this performance-interpretability trade-off is an interesting direction for future research.

Acknowledgments and Disclosure of Funding

This research was supported by the Federal Ministry for Economic Affairs and Climate Action of Germany (BMWK).

References

[1] Aglin, G., Nijssen, S., and Schaus, P. (2020). Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3146–3153.

[2] Aglin, G., Nijssen, S., and Schaus, P. (2022). Pydl8.5. `https://github.com/aia-uclouvain/pydl8.5`. Accessed 13.11.2022.

[3] Barros, R. C., Basgalupp, M. P., De Carvalho, A. C., and Freitas, A. A. (2011). A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(3):291–312.

[4] Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

[5] Bertsimas, D. and Dunn, J. (2017). Optimal classification trees. *Machine Learning*, 106(7):1039–1082.

[6] Blanquero, R., Carrizosa, E., Molero-Río, C., and Morales, D. R. (2020). Sparsity in optimal randomized classification trees. *European Journal of Operational Research*, 284(1):255–272.

[7] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees.* Wadsworth.

[8] Chang, C.-H., Caruana, R., and Goldenberg, A. (2021). Node-gam: Neural generalized additive model for interpretable deep learning. *arXiv preprint arXiv:2106.01613*.

[9] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357.

[10] Demirović, E., Lukina, A., Hebrard, E., Chan, J., Bailey, J., Leckie, C., Ramamohanarao, K., and Stuckey, P. J. (2022). Murtree: Optimal decision trees via dynamic programming and search. *Journal of Machine Learning Research*, 23(26):1–47.

[11] Dua, D. and Graff, C. (2017). UCI machine learning repository.

[12] Freitas, A. A. (2002). *Data mining and knowledge discovery with evolutionary algorithms*. Springer Science & Business Media.

[13] Frosst, N. and Hinton, G. (2017). Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*.

[14] Irsoy, O., Yıldız, O. T., and Alpaydın, E. (2012). Soft decision trees. In *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*, pages 1819–1822. IEEE.

[15] Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and Wilson, A. G. (2018). Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*.

[16] Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

[17] Jordan, M. I. and Jacobs, R. A. (1994). Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214.

[18] Karthikeyan, A., Jain, N., Natarajan, N., and Jain, P. (2022). Learning accurate decision trees with bandit feedback via quantized gradient descent. *Transactions of Machine Learning Research*.

[19] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[20] Kontschieder, P., Fiterau, M., Criminisi, A., and Bulo, S. R. (2015). Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*, pages 1467–1475.

[21] Kuhn, M., Johnson, K., et al. (2013). *Applied predictive modeling*, volume 26. Springer.

[22] Leng, Z., Tan, M., Liu, C., Cubuk, E. D., Shi, X., Cheng, S., and Anguelov, D. (2022). Polyloss: A polynomial expansion perspective of classification loss functions. *arXiv preprint arXiv:2204.12511*.

[23] Lin, J., Zhong, C., Hu, D., Rudin, C., and Seltzer, M. (2020). Generalized and scalable optimal sparse decision trees. In *International Conference on Machine Learning*, pages 6150–6160. PMLR.

[24] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988.

[25] Loh, W.-Y. (2002). Regression tress with unbiased variable selection and interaction detection. *Statistica sinica*, pages 361–386.

[26] Loh, W.-Y. (2009). Improving the precision of classification trees. *The Annals of Applied Statistics*, pages 1710–1737.

[27] Mazumder, R., Meng, X., and Wang, H. (2022). Quant-bnb: A scalable branch-and-bound method for optimal decision trees with continuous features. In *International Conference on Machine Learning*, pages 15255–15277. PMLR.

[28] Molnar, C. (2020). *Interpretable machine learning*. Lulu. com.

[29] Norouzi, M., Collins, M., Johnson, M. A., Fleet, D. J., and Kohli, P. (2015). Efficient non-greedy optimization of decision trees. *Advances in neural information processing systems*, 28.

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

[31] Peters, B., Niculae, V., and Martins, A. F. (2019). Sparse sequence-to-sequence models. *arXiv preprint arXiv:1905.05702*.

[32] Popov, S., Morozov, S., and Babenko, A. (2019). Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*.

[33] Pysiak, K. (2021). Genetictree. `https://github.com/pysiakk/GeneticTree`. Accessed 17.08.2022.

[34] Quinlan, J. R. (1993). *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[35] Tanno, R., Arulkumaran, K., Alexander, D., Criminisi, A., and Nori, A. (2019). Adaptive neural trees. In *International Conference on Machine Learning*, pages 6166–6175. PMLR.

[36] Xu, Z., Zhu, G., Yuan, C., and Huang, Y. (2022). One-stage tree: end-to-end tree builder and pruner. *Machine Learning*, 111(5):1959–1985.

[37] Yang, Y., Morillo, I. G., and Hospedales, T. M. (2018). Deep neural decision trees. *arXiv preprint arXiv:1806.06988*.

[38] Yang, Y., Morillo, I. G., and Hospedales, T. M. (2022). Deep neural decision trees. `https://github.com/wOOL/DNDT`. Accessed 13.11.2022.

[39] Zantedeschi, V., Kusner, M., and Niculae, V. (2021). Learning binary decision trees by argmin differentiation. In *International Conference on Machine Learning*, pages 12298–12309. PMLR.

[40] Zharmagambetov, A., Hada, S. S., Gabidolla, M., and Carreira-Perpinán, M. A. (2021). Non-greedy algorithms for decision tree optimization: An experimental comparison. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

A Gradient Descent Optimization

We use stochastic gradient descent (SGD) to minimize the loss function of GradTree, which is outlined in Algorithm 2. We use backpropagation to calculate the gradients in Line 11-13. Furthermore, our implementation optimizes Algorithm 2 by exploiting common SGD techniques, including mini-batch calculation and momentum using the Adam optimizer [19]. We further apply weight averaging [15] over 5 consecutive checkpoints, similar to Popov et al. [32]. Our novel tree pass function allows formulating Line 7-9 as a single set of matrix operations for an entire batch, which results in a very efficient optimization. Moreover, we implement an early stopping procedure based on the validation loss. To avoid bad initial parametrizations during the initialization, we additionally implement random restarts where the best parameters are selected based on the validation loss.

Algorithm 2 Gradient Descent Training for Decision Trees

```
 1: function TRAINDT($I, T, L, X, \boldsymbol{y}, n, c, d, \xi$)
 2:     $I \sim \mathcal{U}\left(-\sqrt{\frac{6}{2^{2d-1}+n}}, \sqrt{\frac{6}{2^{2d-1}+n}}\right)$
 3:     $T \sim \mathcal{U}\left(-\sqrt{\frac{6}{2^{2d-1}+n}}, \sqrt{\frac{6}{2^{2d-1}+n}}\right)$
 4:     $L \sim \mathcal{U}\left(-\sqrt{\frac{6}{2^{2d}+c}}, \sqrt{\frac{6}{2^{2d}+c}}\right)$
 5:     for $i = 1, \dots, \xi$ do
 6:         $\hat{\boldsymbol{y}} \leftarrow \emptyset$
 7:         for $j = 1, \dots, |X|$ do
 8:             $\hat{y}_j = \text{PASS}(I, T, L, X_j)$
 9:             $\hat{\boldsymbol{y}} \leftarrow \hat{\boldsymbol{y}} \cup \hat{y}_j$
10:         end for
11:         $I \leftarrow I + \eta \frac{\partial}{\partial I}\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$
12:         $T \leftarrow T + \eta \frac{\partial}{\partial T}\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$
13:         $L \leftarrow L + \eta \frac{\partial}{\partial L}\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$
14:     end for
15: end function
```

B Additional Results

Table 3: **Ablation Study Optimized Parameters.** We report macro F1-scores (mean $\pm$ stdev over 10 trials) with optimized parameters. The datasets are sorted by the number of features.

	ST Entmax (ours)	ST Gumbel Softmax	Temperature Annealing
Blood Transfusion	**0.628 ± .036 (1)**	0.482 ± .083 (3)	0.606 ± .073 (2)
Banknote Authentication	**0.987 ± .007 (1)**	0.770 ± .144 (3)	0.946 ± .041 (2)
Titanic	**0.776 ± .025 (1)**	0.543 ± .093 (3)	0.762 ± .036 (2)
Raisins	0.840 ± .022 (2)	0.792 ± .069 (3)	**0.846 ± .028 (1)**
Rice	0.926 ± .007 (2)	0.859 ± .058 (3)	**0.927 ± .006 (1)**
Echocardiogram	**0.658 ± .113 (1)**	0.574 ± .108 (3)	0.619 ± .151 (2)
Wisconsin Breast Cancer	**0.904 ± .022 (1)**	0.844 ± .059 (3)	0.893 ± .031 (2)
Loan House	**0.714 ± .041 (1)**	0.522 ± .098 (3)	0.692 ± .051 (2)
Heart Failure	**0.750 ± .070 (1)**	0.556 ± .095 (3)	0.749 ± .060 (2)
Heart Disease	**0.779 ± .047 (1)**	0.607 ± .136 (3)	0.754 ± .035 (2)
Adult	**0.743 ± .034 (1)**	0.583 ± .029 (3)	0.737 ± .045 (2)
Bank Marketing	**0.640 ± .027 (1)**	0.370 ± .052 (3)	0.611 ± .053 (2)
Congressional Voting	**0.950 ± .021 (1)**	0.710 ± .201 (3)	0.946 ± .026 (2)
Absenteeism	**0.626 ± .047 (1)**	0.489 ± .059 (3)	0.604 ± .036 (2)
Hepatitis	**0.608 ± .078 (1)**	0.395 ± .138 (3)	0.576 ± .122 (2)
German	0.592 ± .068 (2)	0.486 ± .072 (3)	**0.634 ± .028 (1)**
Mushroom	**1.000 ± .001 (1)**	0.682 ± .063 (3)	0.996 ± .005 (2)
Credit Card	**0.674 ± .014 (1)**	0.488 ± .019 (3)	0.668 ± .015 (2)
Horse Colic	0.842 ± .039 (2)	0.442 ± .126 (3)	**0.843 ± .039 (1)**
Thyroid	**0.905 ± .010 (1)**	0.449 ± .087 (3)	0.888 ± .017 (2)
Cervical Cancer	0.521 ± .043 (2)	0.333 ± .191 (3)	**0.523 ± .042 (1)**
Spambase	**0.903 ± .025 (1)**	0.541 ± .093 (3)	0.875 ± .019 (2)
Mean ↑	**0.771 ± .036 (1)**	0.569 ± .094 (3)	0.759 ± .044 (2)
Mean Reciprocal Rank (MRR) ↑	**0.886 ± .214 (1)**	0.333 ± .000 (3)	0.613 ± .214 (2)

Table 4: **Ablation Study Default Parameters.** We report macro F1-scores (mean $\pm$ stdev over 10 trials) with default parameters. The datasets are sorted by the number of features.

	ST Entmax (ours)	ST Gumbel Softmax	Temperature Annealing
Blood Transfusion	**0.627** $\pm$ **.056 (1)**	0.497 $\pm$ .086 (3)	0.618 $\pm$ .045 (2)
Banknote Authentication	**0.980** $\pm$ **.008 (1)**	0.722 $\pm$ .126 (3)	0.970 $\pm$ .007 (2)
Titanic	**0.782** $\pm$ **.035 (1)**	0.623 $\pm$ .096 (3)	0.769 $\pm$ .033 (2)
Raisins	0.850 $\pm$ .025 (2)	0.802 $\pm$ .055 (3)	**0.853** $\pm$ **.016 (1)**
Rice	**0.926** $\pm$ **.006 (1)**	0.802 $\pm$ .101 (3)	**0.926** $\pm$ **.006 (1)**
Echocardiogram	**0.648** $\pm$ **.130 (1)**	0.571 $\pm$ .093 (3)	0.595 $\pm$ .121 (2)
Wisconsin Breast Cancer	0.902 $\pm$ .029 (2)	0.841 $\pm$ .129 (3)	**0.903** $\pm$ **.028 (1)**
Loan House	0.695 $\pm$ .035 (2)	0.490 $\pm$ .113 (3)	**0.706** $\pm$ **.040 (1)**
Heart Failure	0.745 $\pm$ .063 (2)	0.489 $\pm$ .095 (3)	**0.761** $\pm$ **.055 (1)**
Heart Disease	**0.736** $\pm$ **.069 (1)**	0.473 $\pm$ .148 (3)	0.721 $\pm$ .049 (2)
Adult	**0.749** $\pm$ **.028 (1)**	0.601 $\pm$ .034 (3)	0.737 $\pm$ .042 (2)
Bank Marketing	**0.626** $\pm$ **.017 (1)**	0.389 $\pm$ .084 (3)	**0.626** $\pm$ **.010 (1)**
Congressional Voting	**0.953** $\pm$ **.021 (1)**	0.642 $\pm$ .161 (3)	**0.953** $\pm$ **.021 (2)**
Absenteeism	0.620 $\pm$ .050 (2)	0.452 $\pm$ .056 (3)	**0.625** $\pm$ **.065 (1)**
Hepatitis	**0.609** $\pm$ **.103 (1)**	0.390 $\pm$ .125 (3)	0.574 $\pm$ .124 (2)
German	0.584 $\pm$ .057 (2)	0.511 $\pm$ .086 (3)	**0.596** $\pm$ **.052 (1)**
Mushroom	**0.997** $\pm$ **.003 (1)**	0.701 $\pm$ .083 (3)	0.949 $\pm$ .095 (2)
Credit Card	**0.676** $\pm$ **.009 (1)**	0.479 $\pm$ .020 (3)	0.669 $\pm$ .005 (2)
Horse Colic	0.842 $\pm$ .033 (2)	0.458 $\pm$ .101 (3)	**0.857** $\pm$ **.050 (1)**
Thyroid	**0.872** $\pm$ **.015 (1)**	0.457 $\pm$ .099 (3)	0.866 $\pm$ .013 (2)
Cervical Cancer	**0.496** $\pm$ **.047 (1)**	0.354 $\pm$ .160 (3)	0.479 $\pm$ .048 (2)
Spambase	**0.893** $\pm$ **.015 (1)**	0.580 $\pm$ .077 (3)	0.891 $\pm$ .008 (2)
Mean $\uparrow$	**0.764** $\pm$ **.039 (1)**	0.560 $\pm$ .097 (3)	0.757 $\pm$ .042 (2)
Mean Reciprocal Rank (MRR) $\uparrow$	**0.841** $\pm$ **.238 (1)**	0.333 $\pm$ .000 (3)	0.705 $\pm$ .252 (2)

Table 5: **Performance Comparison Default Hyperparameters.** We report macro F1-scores (mean $\pm$ stdev over 10 trials) with default parameters. The datasets are sorted by the number of features.

	Gradient-Based		Non-Greedy		Greedy
	GradTree (ours)	DNDT	GeneticTree	DL8.5 (Optimal)	CART
Blood Transfusion	**0.627** $\pm$ **.056 (1)**	0.558 $\pm$ .059 (5)	0.574 $\pm$ .093 (3)	0.590 $\pm$ .034 (2)	0.573 $\pm$ .036 (4)
Banknote Authentication	0.980 $\pm$ .008 (2)	0.886 $\pm$ .024 (5)	0.928 $\pm$ .022 (4)	0.962 $\pm$ .011 (3)	**0.981** $\pm$ **.006 (1)**
Titanic	**0.782** $\pm$ **.035 (1)**	0.740 $\pm$ .026 (4)	0.763 $\pm$ .041 (2)	0.754 $\pm$ .031 (3)	0.735 $\pm$ .041 (5)
Raisins	0.850 $\pm$ .025 (2)	0.832 $\pm$ .026 (4)	**0.859** $\pm$ **.022 (1)**	0.849 $\pm$ .027 (3)	0.811 $\pm$ .028 (5)
Rice	0.926 $\pm$ .006 (2)	0.902 $\pm$ .018 (5)	**0.927** $\pm$ **.005 (1)**	0.925 $\pm$ .008 (3)	0.905 $\pm$ .010 (4)
Echocardiogram	**0.648** $\pm$ **.130 (1)**	0.543 $\pm$ .117 (5)	0.563 $\pm$ .099 (3)	0.609 $\pm$ .112 (2)	0.559 $\pm$ .075 (4)
Wisconsin Breast Cancer	0.902 $\pm$ .029 (3)	**0.907** $\pm$ **.037 (1)**	0.888 $\pm$ .021 (5)	0.896 $\pm$ .021 (4)	0.904 $\pm$ .025 (2)
Loan House	**0.695** $\pm$ **.035 (1)**	0.475 $\pm$ .043 (4)	0.461 $\pm$ .093 (5)	0.607 $\pm$ .045 (3)	0.671 $\pm$ .056 (2)
Heart Failure	0.745 $\pm$ .063 (2)	0.580 $\pm$ .077 (5)	0.740 $\pm$ .055 (3)	0.692 $\pm$ .062 (4)	**0.755** $\pm$ **.060 (1)**
Heart Disease	**0.736** $\pm$ **.069 (1)**	$n > 12$	0.704 $\pm$ .059 (3)	0.722 $\pm$ .065 (2)	0.670 $\pm$ .090 (4)
Adult	0.749 $\pm$ .028 (2)	$n > 12$	0.478 $\pm$ .068 (4)	0.723 $\pm$ .011 (3)	**0.773** $\pm$ **.008 (1)**
Bank Marketing	**0.626** $\pm$ **.017 (1)**	$n > 12$	0.473 $\pm$ .002 (4)	0.502 $\pm$ .011 (3)	0.616 $\pm$ .007 (2)
Congressional Voting	**0.953** $\pm$ **.021 (1)**	$n > 12$	0.932 $\pm$ .034 (3)	0.924 $\pm$ .043 (4)	0.933 $\pm$ .032 (2)
Absenteeism	**0.620** $\pm$ **.050 (1)**	$n > 12$	0.417 $\pm$ .035 (4)	0.587 $\pm$ .047 (2)	0.580 $\pm$ .045 (3)
Hepatitis	0.609 $\pm$ .103 (2)	$n > 12$	0.486 $\pm$ .074 (4)	0.586 $\pm$ .083 (3)	**0.610** $\pm$ **.123 (1)**
German	0.584 $\pm$ .057 (2)	$n > 12$	0.412 $\pm$ .005 (4)	0.556 $\pm$ .035 (3)	**0.595** $\pm$ **.028 (1)**
Mushroom	0.997 $\pm$ .003 (3)	$n > 12$	0.984 $\pm$ .003 (4)	**0.999** $\pm$ **.001 (1)**	0.999 $\pm$ .001 (2)
Credit Card	0.676 $\pm$ .009 (4)	$n > 12$	**0.685** $\pm$ **.004 (1)**	0.679 $\pm$ .007 (3)	0.679 $\pm$ .007 (2)
Horse Colic	**0.842** $\pm$ **.033 (1)**	$n > 12$	0.794 $\pm$ .042 (2)	0.708 $\pm$ .038 (4)	0.758 $\pm$ .053 (3)
Thyroid	0.872 $\pm$ .015 (2)	$n > 12$	0.476 $\pm$ .101 (4)	0.682 $\pm$ .018 (3)	**0.912** $\pm$ **.013 (1)**
Cervical Cancer	0.496 $\pm$ .047 (3)	$n > 12$	**0.514** $\pm$ **.034 (1)**	0.488 $\pm$ .027 (4)	0.505 $\pm$ .033 (2)
Spambase	0.893 $\pm$ .015 (2)	$n > 12$	0.864 $\pm$ .014 (3)	0.863 $\pm$ .011 (4)	**0.917** $\pm$ **.011 (1)**
Mean $\uparrow$	**0.764** $\pm$ **.144 (1)**	-	0.678 $\pm$ .197 (4)	0.723 $\pm$ .154 (3)	0.747 $\pm$ .153 (2)
Mean Reciprocal Rank (MRR) $\uparrow$	**0.670** $\pm$ **.289 (1)**	0.306 $\pm$ .262 (5)	0.427 $\pm$ .287 (3)	0.371 $\pm$ .164 (4)	0.571 $\pm$ .318 (2)

Table 6: **Train Performance Comparison.** We report macro F1-scores (mean ± stdev over 10 trials) on the training data. The datasets are sorted by the number of features.

	Gradient-Based		Non-Greedy		Greedy
	GradTree (ours)	DNDT	GeneticTree	DL8.5 (Optimal)	CART
Blood Transfusion	0.686 ± .032 (3)	0.552 ± .085 (5)	0.615 ± .122 (4)	0.711 ± .045 (2)	**0.832 ± .040 (1)**
Banknote Authentication	0.995 ± .003 (2)	0.907 ± .012 (5)	0.933 ± .016 (4)	0.967 ± .003 (3)	**0.999 ± .001 (1)**
Titanic	0.761 ± .128 (5)	0.791 ± .025 (4)	0.908 ± .023 (3)	**0.970 ± .006 (1)**	0.949 ± .012 (2)
Raisins	**0.892 ± .024 (1)**	0.829 ± .027 (5)	0.863 ± .004 (4)	0.889 ± .004 (2)	0.866 ± .004 (3)
Rice	0.925 ± .003 (3)	0.913 ± .012 (5)	0.920 ± .002 (4)	**0.930 ± .002 (1)**	0.927 ± .002 (2)
Echocardiogram	0.827 ± .088 (4)	0.817 ± .034 (5)	0.844 ± .032 (3)	0.935 ± .013 (2)	**0.981 ± .009 (1)**
Wisconsin Breast Cancer	0.947 ± .015 (2)	0.936 ± .015 (3)	0.907 ± .009 (5)	**0.964 ± .005 (1)**	0.915 ± .006 (4)
Loan House	0.735 ± .009 (4)	0.712 ± .010 (5)	0.897 ± .012 (3)	**0.969 ± .006 (1)**	0.938 ± .009 (2)
Heart Failure	0.785 ± .038 (4)	0.773 ± .053 (5)	0.795 ± .024 (3)	**0.912 ± .008 (1)**	0.818 ± .021 (2)
Heart Disease	0.851 ± .027 (4)	$n > 12$	0.902 ± .020 (3)	**0.990 ± .005 (1)**	0.964 ± .010 (2)
Adult	0.875 ± .050 (4)	$n > 12$	0.932 ± .013 (3)	0.967 ± .001 (2)	**0.973 ± .001 (1)**
Bank Marketing	0.603 ± .029 (4)	$n > 12$	0.971 ± .003 (3)	0.981 ± .001 (2)	**0.984 ± .001 (1)**
Congressional Voting	0.971 ± .019 (4)	$n > 12$	0.978 ± .005 (3)	1.000 ± .001 (2)	**1.000 ± .000 (1)**
Absenteeism	0.778 ± .050 (4)	$n > 12$	0.842 ± .011 (3)	0.920 ± .009 (2)	**0.952 ± .008 (1)**
Hepatitis	0.931 ± .038 (4)	$n > 12$	0.967 ± .011 (3)	**1.000 ± .000 (1)**	0.998 ± .003 (2)
German	0.589 ± .048 (4)	$n > 12$	0.891 ± .010 (3)	**0.958 ± .002 (1)**	0.940 ± .013 (2)
Mushroom	**1.000 ± .000 (1)**	$n > 12$	0.993 ± .005 (4)	**1.000 ± .000 (1)**	**1.000 ± .000 (1)**
Credit Card	0.651 ± .020 (4)	$n > 12$	0.691 ± .002 (3)	0.710 ± .003 (2)	**0.758 ± .008 (1)**
Horse Colic	0.882 ± .033 (4)	$n > 12$	0.913 ± .019 (3)	**1.000 ± .000 (1)**	0.963 ± .008 (2)
Thyroid	0.914 ± .012 (4)	$n > 12$	0.927 ± .014 (3)	0.937 ± .002 (2)	**0.987 ± .002 (1)**
Cervical Cancer	0.593 ± .128 (4)	$n > 12$	0.691 ± .046 (3)	0.767 ± .020 (2)	**0.925 ± .027 (1)**
Spambase	0.905 ± .020 (2)	$n > 12$	0.858 ± .020 (4)	0.879 ± .002 (3)	**0.965 ± .003 (1)**
Mean ↑	0.823 ± .132 (4)	-	0.874 ± .098 (3)	0.925 ± .087 (2)	**0.938 ± .065 (1)**
Mean Reciprocal Rank (MRR) ↑	0.358 ± .226 (3)	0.220 ± .045 (5)	0.305 ± .044 (4)	0.712 ± .263 (2)	**0.754 ± .282 (1)**

Table 7: **Tree Size Comparison.** We report the average tree size based on the optimized hyperparameters (mean ± stdev over 10 trials). The datasets are sorted by the number of features.

	Gradient-Based		Non-Greedy		Greedy
	GradTree (ours)	DNDT	GeneticTree	DL8.5 (Optimal)	CART
Blood Transfusion	29.200 ± 8.219 (3)	24.000 ± 0.000 (2)	5.200 ± 3.027 (1)	24.000 ± 1.612 (3)	165.600 ± 21.837 (5)
Banknote Authentication	60.000 ± 11.216 (5)	24.000 ± 0.000 (2)	**12.800 ± 5.325 (1)**	26.800 ± 0.600 (3)	44.800 ± 4.686 (4)
Titanic	39.600 ± 10.161 (3)	142.000 ± 0.000 (5)	**10.400 ± 3.105 (1)**	28.200 ± 0.980 (3)	21.600 ± 0.917 (2)
Raisins	114.800 ± 33.686 (4)	142.000 ± 0.000 (5)	3.600 ± 1.800 (2)	29.400 ± 1.497 (3)	**3.000 ± 0.000 (1)**
Rice	41.000 ± 10.040 (4)	142.000 ± 0.000 (5)	3.000 ± 0.000 (2)	30.200 ± 0.980 (3)	**3.000 ± 0.000 (1)**
Echocardiogram	43.200 ± 12.820 (4)	272.000 ± 0.000 (5)	**9.000 ± 3.098 (1)**	30.000 ± 1.342 (2)	36.200 ± 5.810 (3)
Wisconsin Breast Cancer	61.800 ± 13.363 (4)	1,044.000 ± 0.000 (5)	4.000 ± 1.612 (2)	29.400 ± 0.800 (3)	**3.000 ± 0.000 (1)**
Loan House	19.200 ± 5.250 (2)	2,070.000 ± 0.000 (5)	**6.800 ± 2.750 (1)**	28.200 ± 1.327 (4)	26.200 ± 2.040 (3)
Heart Failure	27.200 ± 9.442 (3)	4,120.000 ± 0.000 (5)	**3.200 ± 0.600 (1)**	30.800 ± 0.600 (4)	14.200 ± 0.980 (2)
Heart Disease	36.800 ± 7.454 (4)	$n > 12$	**11.000 ± 4.099 (1)**	27.600 ± 1.562 (3)	24.600 ± 3.072 (2)
Adult	128.000 ± 37.194 (3)	$n > 12$	**9.400 ± 4.964 (1)**	25.200 ± 0.600 (2)	156.600 ± 23.079 (4)
Bank Marketing	6.800 ± 3.027 (2)	$n > 12$	**3.400 ± 1.200 (1)**	27.200 ± 2.272 (3)	110.600 ± 10.500 (4)
Congressional Voting	5.600 ± 2.375 (2)	$n > 12$	**5.600 ± 0.917 (1)**	20.400 ± 6.696 (4)	19.200 ± 6.416 (3)
Absenteeism	148.800 ± 18.187 (4)	$n > 12$	**7.800 ± 0.980 (1)**	30.600 ± 0.800 (2)	43.200 ± 2.750 (3)
Hepatitis	12.200 ± 3.600 (2)	$n > 12$	**5.800 ± 2.713 (1)**	14.800 ± 2.088 (4)	12.600 ± 1.744 (3)
German	27.600 ± 4.104 (2)	$n > 12$	**6.200 ± 0.980 (1)**	30.400 ± 0.917 (3)	34.600 ± 5.713 (4)
Mushroom	25.600 ± 5.731 (4)	$n > 12$	**5.200 ± 1.661 (1)**	21.200 ± 2.750 (3)	14.000 ± 1.342 (2)
Credit Card	92.600 ± 39.636 (3)	$n > 12$	**3.000 ± 0.000 (1)**	31.000 ± 0.000 (2)	354.800 ± 31.603 (4)
Horse Colic	22.400 ± 4.737 (3)	$n > 12$	**4.000 ± 1.612 (1)**	29.400 ± 1.200 (4)	12.600 ± 1.200 (2)
Thyroid	96.800 ± 16.863 (4)	$n > 12$	**5.600 ± 1.800 (1)**	30.600 ± 0.800 (2)	66.400 ± 4.652 (3)
Cervical Cancer	34.000 ± 23.669 (3)	$n > 12$	**14.000 ± 7.550 (1)**	30.800 ± 0.600 (2)	99.000 ± 8.532 (4)
Spambase	120.200 ± 37.010 (3)	$n > 12$	**12.400 ± 5.800 (1)**	29.600 ± 0.917 (2)	201.000 ± 14.642 (4)
Mean ↑	54.245 ± 42.824 (3)	886.667 ± 1,387.861 (5)	**6.882 ± 3.457 (1)**	27.536 ± 4.146 (2)	66.673 ± 86.224 (4)
Mean Reciprocal Rank (MRR) ↑	0.331 ± 0.102 (4)	0.267 ± 0.132 (5)	**0.932 ± 0.176 (1)**	0.367 ± 0.098 (3)	0.430 ± 0.252 (2)

Table 8: **Runtime Comparison.** We report runtime without restarts based on the optimized hyperparameters (mean ± stdev over 10 trials). The datasets are sorted by the number of features.

	Gradient-Based		Non-Greedy		Greedy
	GradTree (ours)	DNDT	GeneticTree	DL8.5 (Optimal)	CART
Blood Transfusion	13.007 ± 1.000 (5)	3.704 ± 1.000 (3)	10.514 ± 4.000 (4)	0.031 ± 0.000 (2)	**0.002** ± **0.000 (1)**
Banknote Authentication	13.209 ± 2.000 (4)	34.425 ± 1.000 (5)	10.318 ± 3.000 (3)	0.025 ± 0.000 (2)	**0.003** ± **0.000 (1)**
Titanic	29.579 ± 1.000 (5)	6.815 ± 0.000 (4)	3.677 ± 1.000 (3)	0.259 ± 0.000 (2)	**0.002** ± **0.000 (1)**
Raisins	30.611 ± 2.000 (5)	8.495 ± 1.000 (4)	0.619 ± 0.000 (3)	0.198 ± 0.000 (2)	**0.004** ± **0.000 (1)**
Rice	21.487 ± 4.000 (5)	11.955 ± 1.000 (4)	1.329 ± 0.000 (3)	0.475 ± 0.000 (2)	**0.013** ± **0.000 (1)**
Echocardiogram	9.038 ± 1.000 (4)	9.564 ± 2.000 (5)	0.304 ± 0.000 (3)	0.093 ± 0.000 (2)	**0.002** ± **0.000 (1)**
Wisconsin Breast Cancer	29.246 ± 2.000 (5)	6.837 ± 1.000 (4)	1.187 ± 0.000 (3)	0.461 ± 0.000 (2)	**0.003** ± **0.000 (1)**
Loan House	10.798 ± 1.000 (4)	52.793 ± 7.000 (5)	6.751 ± 2.000 (3)	0.524 ± 0.000 (2)	**0.002** ± **0.000 (1)**
Heart Failure	28.395 ± 1.000 (5)	23.747 ± 10.000 (4)	1.473 ± 0.000 (3)	0.201 ± 0.000 (2)	**0.002** ± **0.000 (1)**
Heart Disease	15.565 ± 2.000 (4)	$n > 12$	4.446 ± 1.000 (3)	0.636 ± 0.000 (2)	**0.002** ± **0.000 (1)**
Adult	86.241 ± 12.000 (4)	$n > 12$	26.012 ± 7.000 (3)	22.321 ± 1.000 (2)	**0.071** ± **0.000 (1)**
Bank Marketing	153.494 ± 38.000 (4)	$n > 12$	129.428 ± 8.000 (3)	54.545 ± 1.000 (2)	**0.072** ± **0.000 (1)**
Congressional Voting	32.935 ± 3.000 (4)	$n > 12$	6.503 ± 1.000 (3)	4.100 ± 7.000 (2)	**0.002** ± **0.000 (1)**
Absenteeism	28.941 ± 1.000 (4)	$n > 12$	14.045 ± 3.000 (3)	3.893 ± 0.000 (2)	**0.003** ± **0.000 (1)**
Hepatitis	28.427 ± 1.000 (4)	$n > 12$	2.273 ± 1.000 (3)	0.023 ± 0.000 (2)	**0.002** ± **0.000 (1)**
German	12.307 ± 2.000 (4)	$n > 12$	3.141 ± 1.000 (2)	10.288 ± 0.000 (3)	**0.003** ± **0.000 (1)**
Mushroom	58.810 ± 70.000 (4)	$n > 12$	12.507 ± 2.000 (3)	0.507 ± 0.000 (2)	**0.007** ± **0.000 (1)**
Credit Card	111.425 ± 36.000 (3)	$n > 12$	4.416 ± 0.000 (2)	298.572 ± 6.000 (4)	**0.351** ± **0.000 (1)**
Horse Colic	9.811 ± 1.000 (4)	$n > 12$	0.316 ± 0.000 (2)	2.365 ± 2.000 (3)	**0.002** ± **0.000 (1)**
Thyroid	35.965 ± 13.000 (2)	$n > 12$	38.287 ± 15.000 (3)	109.123 ± 3.000 (4)	**0.013** ± **0.000 (1)**
Cervical Cancer	13.465 ± 2.000 (4)	$n > 12$	2.760 ± 1.000 (3)	0.126 ± 0.000 (2)	**0.006** ± **0.000 (1)**
Spambase	41.983 ± 5.000 (4)	$n > 12$	19.289 ± 6.000 (3)	4.576 ± 0.000 (2)	**0.036** ± **0.000 (1)**
Mean ↑	44.745 ± 38.655 (4)	-	13.618 ± 27.539 (2)	23.334 ± 66.445 (3)	**0.027** ± **0.075 (1)**
Mean Reciprocal Rank (MRR) ↑	0.244 ± 0.037 (4)	0.243 ± 0.042 (5)	0.352 ± 0.063 (3)	0.462 ± 0.084 (2)	**1.000** ± **0.000 (1)**

C Hyperparameters

In the following, we report the hyperparameters used for each approach. The hyperparameters were selected based on a random search over a predefined parameter range for GradTree, CART and GeneticTree and are summarized in Table 10 to Table 12. All parameters that were considered are noted in the tables. The number of trials (300) was equal for each approach. For GradTree and DNDT, we did not optimize the batch size as well as the number of epochs, but used early stopping with a predefined patience of 200. Additionally, we used 3 random restarts to prevent bad initial parametrizations and selected the best model based on the validation loss. A gradient-based optimization allows using an arbitrary loss function for the optimization. During preliminary experiments, we observed that it is beneficial to adjust the loss function for specific datasets. More specifically, we allowed adjusting the cross-entropy loss by adding a focal factor [24] of 3. Additionally, we considered using PolyLoss [22] within the HPO to tailor the loss function for the specific task. For DL8.5 the relevant tunable hyperparameters according to the authors are the maximum depth and the minimum support. However, the maximum depth strongly impacts the runtime, which is why we fixed the maximum depth to 4, similar to the maximum depth used during the experiments of Demirović et al. [10] and Aglin et al. [1]. Running the experiments with a higher depth becomes infeasible for many datasets. In preliminary experiments, we also observed that changing the depth has no positive impact on the performance. Furthermore, to assure a fair comparison, we fixed the minimum support to 1 which is equal to the pruning of GradTree. Additionally, we observed in our preliminary experiments, that increasing the minimum support reduces overfitting, but has no positive impact on the test performance (i.e. the train performance decreases, but the test performance is not improved). For DNDT, the number of cut points is the tunable hyperparameter of the model, according to Yang et al. [37]. However, it has to be restricted to 1 in order to generate binary trees for comparability reasons. Therefore, we only optimized the temperature and the learning rate. Furthermore, we extended their implementation to use early stopping based on the validation loss, similar to GradTree, to reduce the runtime.

Table 9: **DNDT Hyperparameters**

Dataset Name	learning_rate	temperature	num_cut
Blood Transfusion	0.050	0.100	1
Banknote Authentication	0.001	0.100	1
Titanic	0.050	0.001	1
Raisins	0.050	0.001	1
Rice	0.100	0.010	1
Echocardiogram	0.005	1.000	1
Wisconsin Breast Cancer	0.100	0.010	1
Loan House	0.001	1.000	1
Heart Failure	0.005	1.000	1

Table 10: **GradTree Hyperparameters**

Dataset Name	depth	lr_index	lr_values	lr_leaf	loss	polyLoss	polyLossEpsilon
Blood Transfusion	8	0.010	0.100	0.010	crossentropy	False	2
Banknote Authentication	7	0.050	0.050	0.100	focal_crossentropy	True	2
Titanic	10	0.005	0.010	0.010	crossentropy	False	2
Raisins	10	0.005	0.005	0.100	crossentropy	True	5
Rice	7	0.050	0.010	0.010	crossentropy	False	2
Echocardiogram	8	0.010	0.050	0.100	crossentropy	True	5
Wisconsin Breast Cancer	10	0.050	0.010	0.100	crossentropy	True	2
Loan House	8	0.005	0.100	0.010	focal_crossentropy	False	2
Heart Failure	10	0.005	0.250	0.100	crossentropy	False	2
Heart Disease	9	0.010	0.050	0.005	focal_crossentropy	False	2
Adult	8	0.050	0.005	0.050	crossentropy	True	5
Bank Marketing	8	0.250	0.250	0.050	crossentropy	False	2
Cervical Cancer	8	0.005	0.010	0.100	crossentropy	True	2
Congressional Voting	10	0.005	0.050	0.010	focal_crossentropy	True	5
Absenteeism	10	0.050	0.010	0.050	focal_crossentropy	True	5
Hepatitis	10	0.005	0.050	0.010	focal_crossentropy	True	5
German	7	0.005	0.050	0.010	crossentropy	True	2
Mushroom	9	0.010	0.010	0.050	focal_crossentropy	False	2
Credit Card	8	0.050	0.100	0.010	focal_crossentropy	False	2
Horse Colic	8	0.250	0.250	0.010	focal_crossentropy	False	2
Thyroid	8	0.010	0.010	0.050	crossentropy	False	2
Spambase	10	0.005	0.010	0.010	crossentropy	False	2

Table 11: **GeneticTree Hyperparameters**

Dataset Name	n_thresholds	n_trees	max_iter	cross_prob	mutation_prob
Blood Transfusion	10	500	500	1.0	0.2
Banknote Authentication	10	500	500	0.8	0.6
Titanic	10	500	250	0.2	0.3
Raisins	10	100	50	1.0	0.6
Rice	10	100	250	0.4	0.3
Echocardiogram	10	50	100	0.8	0.4
Wisconsin Breast Cancer	10	250	100	0.2	0.7
Loan House	10	500	500	1.0	0.2
Heart Failure	10	500	50	0.4	0.6
Heart Disease	10	400	500	0.6	0.4
Adult	10	100	500	0.6	0.6
Bank Marketing	10	500	250	0.8	0.6
Cervical Cancer	10	100	500	0.4	0.9
Congressional Voting	10	500	500	1.0	0.7
Absenteeism	10	500	500	1.0	0.7
Hepatitis	10	500	250	0.2	0.3
German	10	250	500	0.4	0.8
Mushroom	10	400	500	0.6	0.4
Credit Card	10	50	50	0.4	0.1
Horse Colic	10	50	100	0.8	0.4
Thyroid	10	500	500	1.0	0.7
Spambase	10	250	500	0.8	0.6

Table 12: **CART Hyperparameters**

Dataset Name	max_depth	criterion	max_features	min_samples_leaf	min_samples_split	ccp_alpha
Blood Transfusion	9	entropy	None	1	5	0.0
Banknote Authentication	9	gini	None	1	5	0.0
Titanic	7	entropy	None	1	50	0.0
Raisins	8	gini	None	5	2	0.2
Rice	8	gini	None	5	2	0.2
Echocardiogram	9	entropy	None	1	5	0.0
Wisconsin Breast Cancer	7	entropy	None	5	2	0.4
Loan House	10	entropy	None	10	2	0.0
Heart Failure	9	gini	None	5	50	0.0
Heart Disease	8	entropy	None	5	10	0.0
Adult	10	entropy	None	10	2	0.0
Bank Marketing	8	entropy	None	5	10	0.0
Cervical Cancer	9	entropy	None	1	5	0.0
Congressional Voting	10	gini	None	1	2	0.0
Absenteeism	7	entropy	None	1	10	0.0
Hepatitis	9	entropy	None	1	5	0.0
German	7	entropy	None	1	10	0.0
Mushroom	9	entropy	None	1	5	0.0
Credit Card	9	gini	None	1	5	0.0
Horse Colic	10	entropy	None	10	2	0.0
Thyroid	10	entropy	None	10	2	0.0
Spambase	10	gini	None	1	2	0.0

D Datasets

The datasets along with their specifications and source are summarized in Table 13. For all datasets, we performed a standard preprocessing: We applied Leave-one-out encoding to all categorical features. Similar to Popov et al. [32], we further perform a quantile transform, making each feature follows a normal distribution. We use a random $80\%/20\%$ train-test split for all datasets. To account for class imbalance, we rebalanced the training data using SMOTE [9] when the minority class accounts for less than 25% of the data points. Since GradTree and DNDT require a validation set for early stopping, we performed another $80\%/20\%$ split on the training data for those approaches. The remainder of the approaches utilize the complete training data. For DL8.5 additional preprocessing was necessary since they can only handle binary features. Therefore, we one-hot encoded all categorical features and discretized numeric features by one-hot encoding them using quantile binning with 5 bins.

Table 13: **Dataset Specifications.**

Dataset Name	**Number of Features**	**Number of Samples**	**Fraction of Minority Class**	**Source**
Blood Transfusion	4	748	0.238	`https://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center`
Banknote Authentication	4	1,372	0.445	`https://archive.ics.uci.edu/ml/datasets/banknote+authentication`
Titanic	7	891	0.384	`https://www.kaggle.com/c/titanic`
Raisin	7	900	0.500	`https://archive.ics.uci.edu/ml/datasets/Raisin+Dataset`
Rice	7	3,810	0.428	`https://archive.ics.uci.edu/ml/datasets/Rice+%28Cammeo+and+Osmancik%29`
Echocardiogram	8	132	0.189	`https://archive.ics.uci.edu/ml/datasets/echocardiogram`
Wisconsin Breast Cancer	10	569	0.373	`https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)`
Loan House	11	614	0.313	`https://www.kaggle.com/code/sazid28/home-loan-prediction/data`
Heart Failure	12	299	0.321	`https://archive.ics.uci.edu/ml/datasets/Heart+failure+clinical+records`
Heart Disease	13	303	0.459	`https://archive.ics.uci.edu/ml/datasets/heart+disease`
Adult	14	32,561	0.241	`https://archive.ics.uci.edu/ml/datasets/adult`
Bank Marketing	14	45,211	0.117	`https://archive.ics.uci.edu/ml/datasets/bank+marketing`
Congressional Voting	16	435	0.386	`https://archive.ics.uci.edu/ml/datasets/congressional+voting+records`
Absenteeism	18	740	0.377	`https://archive.ics.uci.edu/ml/datasets/Absenteeism+at+work`
Hepatitis	19	155	0.206	`https://archive.ics.uci.edu/ml/datasets/hepatitis`
German	20	1,000	0.300	`https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)`
Mushrooms	22	8,124	0.482	`https://archive.ics.uci.edu/ml/datasets/mushroom`
Credit Card	23	30,000	0.221	`https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients`
Horse Colic	26	368	0.370	`https://archive.ics.uci.edu/ml/datasets/Horse+Colic`
Thyroid	29	9,172	0.262	`https://archive.ics.uci.edu/ml/datasets/thyroid+disease`
Cervical Cancer	31	858	0.064	`https://archive.ics.uci.edu/ml/datasets/Cervical+cancer+%28Risk+Factors%29`
Spambase	57	4,601	0.394	`https://archive.ics.uci.edu/ml/datasets/spambase`