A NEW CLASS OF INTERPRETABLE MODELS: THE GAM TREE

Anonymous authors
Paper under double-blind review

ABSTRACT

The space of interpretable models consists mostly of linear models and their generalizations: decision trees and generalized additive models (GAMs). We propose a further, natural generalization that so far has not been fully explored: the GAM tree, a decision tree having a GAM in each leaf. The GAM tree inherits the benefits of both trees and GAMs and alleviates some of their limitations. Through the hierarchical structure of the tree, it preserves the ability to explain decisions and to segment the data into populations. And, while a single GAM imposes a univariate function of each feature globally, the leaf GAMs make such functions locally adaptive to each population. Crucial for the success of the GAM tree is to learn the tree and the GAMs jointly in a scalable way, which we achieve using a variation of tree alternating optimization. In our experiments with classification and regression, GAM trees consistently exceed both trees and GAMs in accuracy, as expected, but the improvement is often surprisingly large. Further, the entire set of GAMs can be visualized in a single plot, which gives a global picture of the model and at the same time contrasts the differences across populations.

1 INTRODUCTION

Interpretable models in machine learning are, by and large, of the following types: linear models, decision trees and generalized additive models (GAMs). *Linear models* are defined by a linear combination (l.c.) of the input features, and one can simply inspect the sign and magnitude of the coefficient of each feature. *GAMs* generalize linear models to a sum of single-feature ("shape") functions, which can be plotted to inspect the effect of each feature on the output. The shape functions can be quite general, but they are often as simple as piecewise constant (as in a scorecard). One can also consider shape functions of two (or more) features, but this quickly increases the complexity. *Decision trees* are based on a recursive, if-then-else application of a linear model. At each node they can make a decision based on a l.c. of a single feature (axis-aligned), a few features (sparse oblique) or all features (oblique). At a leaf they make a prediction, typically constant[1]. One can understand the predictions of a tree by inspecting its structure and the features used at each node. In all of these models, interpretability is also a matter of degree: a GAM with many features, or a tree with many nodes and/or features in each node becomes harder to understand. Also, axis-aligned trees use very simple nodes but often need larger trees, while (sparse) oblique trees have more complex nodes but smaller trees and are typically much more accurate.

We propose a natural generalization of GAMs and trees, the *GAM tree*. This is a decision tree where each leaf predictor is a separate GAM and the decision nodes can be axis-aligned or (sparse) oblique. Equivalently, it is a set of local GAMs arranged in a hierarchical partition of the space. There are several reasons why this makes sense.

First, *trees and GAMs make strong and very different, complementary modeling assumptions*[2]. A tree partitions the input feature space into boxes or polytopes. This can create complex partitions and

[1]Throughout the paper we use the term "decision tree" to mean trees with constant-label leaves, which is by far the more common case (whether the decision nodes are axis-aligned or oblique). We discuss the case of trees with linear leaf predictors in section 2.

[2]If we have a single feature, then a tree is a particular case of a GAM (a function that is constant on intervals). With more than one feature, the set of functions that are both trees and GAMs is very small and consists of pretty trivial functions (constant over a Cartesian product grid of the space). Most GAMs are not decision trees and vice versa.

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.

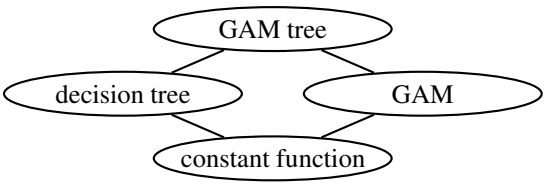


Figure 1: A Hasse diagram showing the partial order (using set inclusion) of the function classes defined by GAM trees, decision trees, GAMs (which also contain linear models) and the constant function.

introduce interactions between features, particularly with oblique splits. However, using a constant label within each region produces a piecewise-constant function. This is well suited for classification, but less so for continuously varying functions, as is usually the case in regression. A GAM can use (in principle) any type of shape function (continuous or not), but the addition of these univariate functions imposes a grid-like structure in the input space and cannot model interactions between multiple features.

This argument suggests that a GAM tree can benefit from the complementary properties of trees and GAMs. Indeed, GAM trees contain both of them as particular cases: a GAM is a GAM tree of depth 0 and a decision tree is a GAM tree having a constant GAM at each leaf. A constant function is the simplest case of both a GAM and a tree. This is illustrated in fig. 1. However, *GAM trees can model a much larger set of functions than either GAMs or trees*.

Second, *GAM trees are significantly more accurate than trees or GAMs but remain highly interpretable*. This is best seen in our experiments, but it can be justified as follows. The improved accuracy of the GAM tree follows from the argument in the previous paragraph. The claim of interpretability needs some explanation. After all, even straighforward combinations of simple models can quickly lose interpretability. For example, nesting linear functions (with an elementwise nonlinearity) results in a multilayer perceptron, and adding decision trees results in a forest, both of which are much more flexible functions but also much harder to interpret. In contrast, we can interpret a GAM tree in two ways, which correspond to the information provided by the tree and by the GAMs:

- By inspecting the tree structure and the root-leaf paths (just as we would do in a regular decision tree), we can understand why a particular instance reaches a particular leaf and what features control this. (With sparse oblique trees, a sparse l.c. can often be understood as a new, "learned" feature.) This also provides an understandable segmentation of the data into populations (one per leaf) and has long been a valued characteristic of decision trees.
- The leaf GAMs information can be visualized as follows: we can plot one graph per feature (just as one would do with a regular GAM), but plotting *all* the leaf GAM functions for a given feature together (since they are univariate functions anyway). This brings extra benefits: we can often see at a glance *global patterns*, such as monotonicity of a given feature (when all the leaf shape functions are increasing, for example); *local, within-population patterns*, by observing the shape functions within a leaf to understand the effect of each feature within one population (as in regular GAMs); and *contrastive, across-population patterns*, by observing the shape functions of all leaves for a given feature in order to understand how the effect of that feature varies depending on the population.

Finally, by making the leaves more powerful predictors, *the GAM tree becomes much smaller than a regular decision tree* (shallower and with fewer nodes). And, by splitting the space into populations, *the leaf GAMs may ignore some features* (that are not needed in a particular leaf)[3]. All of this helps make GAM trees more interpretable.

Why have GAM trees not been explored earlier? We suspect this is due to the difficulty of training tree-structured models, which is NP-hard even in the simplest case (axis-aligned trees with binary features and output; Hyafil & Rivest, 1976). However, the recent strategy of tree alternating optimization (Carreira-Perpiñán & Tavallali, 2018) makes this possible, as long as we can (approximately) solve the optimization over a single decision node or leaf. As we show later, this means we can use

[3]While a single GAM trained on the global population may typically need all features for accurate prediction, each leaf GAM makes predictions for a local population only and may not need all features. This is potentially a major advantage of GAM trees, and we observe it in some of our experiments. However, properly enforcing sparsity over the leaf GAMs means we have to learn sparse GAMs in the first place (i.e., do feature selection on GAMs), which appears not to have been widely studied (Ravikumar et al., 2009; Marra & Wood, 2011). We relegate this study to a future work.

almost any type of GAM in the leaves (although, in our experiments, we focus on piecewise-constant GAMs).

In summary, we think that GAM trees occupy an unexplored, useful area in the accuracy-interpretability space. In the rest of the paper, we place GAM trees within the context of existing work (section 2), describe the GAM tree model precisely (section 3), give a training algorithm (section 4), and demonstrate its performance in classification and regression experiments (section 5).

2 Related work

Decision trees have existed since the 1950s (Hastie et al., 2009; Loh, 2014; Rokach & Maimon, 2015). From a model point of view, the most common ones use axis-aligned splits (Breiman et al., 1984; Quinlan, 1993), but there also exist oblique splits (Breiman et al., 1984), sparse oblique splits (Carreira-Perpiñán & Tavallali, 2018) and bivariate splits (Lubinsky, 1994; Kairgeldin & Carreira-Perpiñán, 2024); the leaf models are typically constant (we discuss linear leaves below). Training decision trees has traditionally used greedy recursive partitioning, which is simple but suboptimal (Hastie et al., 2009). Tree Alternating Optimization (Carreira-Perpiñán & Tavallali, 2018; Zharmagambetov & Carreira-Perpiñán, 2020) finds much better trees by fixing the tree structure and optimizing one node at a time, and it scales to large datasets and trees. Another line of work uses an exhaustive search via mixed-integer optimization (Bertsimas & Dunn, 2017) or dynamic programming (Demirović et al., 2022), but this is impractical beyond tiny trees and datasets, and imposes strong restrictions on the tree (typically axis-aligned, constant leaves and binary features).

Generalized additive models (GAMs; Hastie & Tibshirani, 1990; Wood, 2017) are based on the assumption of a sum of univariate "shape" functions. The shape functions can take many forms (nonparametric smoothers (Hastie & Tibshirani, 1990), splines (Wood, 2017), neural nets, decision forests (Lou et al., 2012), etc.), although most of these tend to perform similarly in accuracy (Chang et al., 2021). Training is often done by alternating optimization over each shape function (called "backfitting" in Hastie & Tibshirani (1990)), but it may be done by gradient descent if the model is differentiable, or in other ways (Lou et al., 2012; Liu et al., 2022). GAMs can be extended to consider pairwise interactions (Hastie & Tibshirani, 1990; Lou et al., 2013). This produces more accurate models but hurts interpretability, because understanding 2D heat maps (as opposed to 1D shape functions) is harder, and there can be many heat maps even with few features.

Hybrids of trees and GAMs. As far as we know, we are the first to describe the GAM tree as a general model, to give an algorithm to optimize all its parameters jointly and to study its interpretability. However, there do exist previous works that considered some special cases of GAM trees. The most prominent one is a tree with linear leaves. This was already described as a variant of regular trees for regression in the foundational work of CART (Breiman et al., 1984), which only mentions them in passing in p. 248, and C4.5 (Quinlan, 1992), which called them "model trees" and "M5" but decided not to include them in the C4.5 suite of programs (Quinlan, 1993, section 11.1). To learn such a tree, one grows a regular regression tree with constant-label leaves and then replaces each leaf label with a linear regressor (a few variations of this have been proposed; Zeileis et al., 2008; Loh, 2014; Witten et al., 2016). This procedure is heavily suboptimal, for several reasons. The split criterion used is the variance, or regression squared error as if the decision node's children were two constant-label leaves. This (already in regular trees) is not true for many nodes because their children are decision nodes and the leaves can be several levels down. But it is worse with linear leaves, which are very different from constant-label leaves; the tree structure induction entirely ignores this. Besides, pruning becomes more complex. This produces trees that are overly large and can overfit easily (Loh, 2014). As a result, such trees have been rarely used in practice; the well-known textbook of Hastie et al. (2009) does not even mention them.

The idea has also been used for classification by placing a logistic regression (Chan & Loh, 2004; Landwehr et al., 2005; Witten et al., 2016) or naive Bayes (Kohavi, 1996) model at each leaf. These may be considered special kinds of a GAM using a certain link function. The same can be said (for regression) of the Single-Index Model Tree of Sudjianto et al. (2023). This uses in each leaf a single-index model, which is a nonlinearity applied to a linear combination. All of these works use an axis-aligned decision tree structure grown with greedy recursive partitioning. They thus share the problems mentioned above.

Tree alternating optimization was more recently used to train linear-leaf trees jointly over all its parameters (decision nodes and leaf linear regressors; Zharmagambetov & Carreira-Perpiñán, 2020)

and was shown to achieve much higher accuracy while reducing the tree size considerably. However, this work did not consider using more general GAMs in the leaves.

3 DEFINITION OF THE GAM TREE MODEL

Let $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ be a training set of N (instance, target) pairs, where $\mathbf{x}_n \in \mathbb{R}^D$ is a D-dimensional feature vector, $y_n \in \mathbb{R}$ for regression, and $y_n \in \{0, 1\}$ for classification. Let $T(\mathbf{x}; \boldsymbol{\Theta})$ denote a binary decision tree that outputs a prediction for an input $\mathbf{x}$ by traversing from the root to a single leaf, where a predictive function, GAM, is applied. Each node j, whether a decision node or a leaf, has associated learnable parameters $\boldsymbol{\theta}_j$, and the overall parameter set for the tree is $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_j\}_{j \in \{\mathcal{D} \cup \mathcal{L}\}}$, where $\mathcal{D}$ is the set of all decision nodes, and $\mathcal{L}$ is the set of all leaf nodes. Each internal node j defines a decision function $f_j(\mathbf{x}; \boldsymbol{\theta}_j)$: $\mathbb{R}^D \to \{\texttt{left}(j), \texttt{right}(j)\} \subset \{\mathcal{D} \cup \mathcal{L}\}$, which directs the input $\mathbf{x}$ to one of its two children. We consider (sparse) oblique decision splits: "go to the $\texttt{right}(j)$ if $\mathbf{w}_j^T \mathbf{x} + w_{j0} > 0$, else go to the $\texttt{left}(j)$", with parameters $\boldsymbol{\theta}_j = \{\mathbf{w}_j, w_{j0}\}$. Axis-aligned splits are a special case of this, where $\mathbf{w}_j$ is a one-hot encoded representation of the feature index, and w_{j0} is a splitting threshold.

Now, specifically in GAM trees, each leaf node j uses a prediction function $h_j(\mathbf{x}; \boldsymbol{\theta}_j)$: $\mathbb{R}^D \to \mathbb{R}$ that takes the form:

$$h_j(\mathbf{x}; \boldsymbol{\theta}_j) = h_j^0 + \textstyle\sum_{d=1}^D h_j^d(x_d) \tag{1}$$

where h_j^0 is a constant bias term, and $\{h_j^d(\cdot)\}_{j=1}^D$ are the individual shape functions per each feature d. For regression problems, $h_j(\mathbf{x}; \boldsymbol{\theta}_j)$ is the final output of the model, but for classification there is an additional link function, sigmoid $\sigma(\cdot)$, to convert the raw score into a probability $\sigma(h_j(\mathbf{x}; \boldsymbol{\theta}_j))$. As regards with the specific form of the $h_j(\mathbf{x}; \boldsymbol{\theta}_j)$, in general, we can use any GAM implementation at the leaves including traditional cubic splines and gradient boosting decision trees. In this work, we focus on GAMs expressed as an ensemble of S decision stumps: $h_j(\mathbf{x}; \boldsymbol{\theta}_j) = h_j^0 + \sum_{s=1}^S g(\mathbf{x}; \boldsymbol{\theta}_j^s)$, where each $g(\mathbf{x}; \boldsymbol{\theta}_j^s)$ is a decision stump, that is, an axis-aligned depth-1 tree, that applies a threshold to a single input feature and returns one of two constant values. Because multiple stumps can use the same feature $\phi \in \{1, \ldots, D\}$, their outputs can be aggregated to form the shape function corresponding to feature ϕ. As a result, the stump ensemble defines a valid GAM structure, and can be trained efficiently with the well-established gradient boosting framework, which is known for producing accurate GAMs very efficiently (Chang et al., 2021).

4 AN ALGORITHM TO TRAIN A GAM TREE

If one were to apply the traditional greedy recursive partitioning strategy (as in CART), the resulting tree would be overly large and suboptimal. This is because the split criterion (say, Gini index) at a particular decision node has a hard time guessing what a good partition would be for the eventual leaf GAMs, which can be several levels down. To learn a good GAM tree, it is critical to optimize it jointly over all its parameters (in the decision nodes and in the leaf GAMs), so that the model assumption that the leaf GAMs make (additive feature effects) informs the decision nodes of what a good partition should be.

We apply the strategy of tree alternating optimization of Carreira-Perpiñán & Tavallali (2018). The idea is simple: we pick a fixed parametric form for the tree (say, complete of depth Δ) and optimize the objective function in turn over the parameters of each node given that all other nodes' parameters are kept fixed, iterating until convergence. The success of this relies on two theorems. The *separability condition* states that nodes that are not descendant of each other (e.g. all nodes at the same depth) can be optimized in parallel, each on its reduced set (the instances that reach that node under the current tree parameters). This brings deep decreases of the objective function and parallelism in the computation. The *reduced problem theorem* shows that the problem of optimizing over a specific node takes a special form, which depends on the type of decision node (axis-aligned or sparse oblique in our case) and leaf (GAM in our case). If this can be (approximately) solved, we will reduce the objective function at each iteration (consisting of a reverse breadth-first pass over all nodes). Next, we apply this to our case and give the precise form of the algorithm.

4.1 OBJECTIVE FUNCTION AND OPTIMIZATION

Analogous to training neural networks, where the architecture and initial weights are specified in advance, we begin by defining the initial structure and parameters of the GAM tree $T(\mathbf{x}; \boldsymbol{\Theta})$. We

consider complete tree structures of depth Δ with random initial weights, and treat depth Δ as a hyperparameter. Once the structure determined, we optimize the following objective function:

$$E(\mathbf{\Theta}) = \sum_{n=1}^{N} L(\mathbf{y}_n, T(\mathbf{x}_n; \mathbf{\Theta})) + \lambda \sum_{j \in \mathcal{D}} \|\boldsymbol{\theta}_j\|_1, \tag{2}$$

where $L(\cdot, \cdot)$ denotes the loss function: squared error for regression tasks and cross-entropy for classification. The second term in the objective is an ℓ_1-penalty applied to the parameters of the decision nodes, promoting sparsity via the regularization coefficient $\lambda \geq 0$. When λ is sufficiently large, the optimization may drive all weights at a decision node to zero, effectively disabling the split at that node. This results in implicit pruning, leading to a smaller final model.

Optimization algorithm. Eq. 2 defines a difficult, non-differentiable learning problem where standard gradient-based optimization is not applicable. To address this, we use coordinate descent (i.e., alternating optimization), updating the parameters of one node at a time. This approach is enabled by two key properties of the objective function 2 and the tree structure: (1) the objective decomposes across specific subsets of nodes, and (2) each node's parameters can be optimized via a simplified, self-contained subproblem. First, the objective function is *separable across any subset of non-descendant nodes* (for instance, all nodes at a given depth), due to the hard (non-probabilistic) routing decisions in the tree. As a result, multiple such nodes can be optimized in parallel. Second, for a given node j, the optimization reduces to a *node-specific subproblem* defined over the training instances that currently reach that node, referred to as the *reduced set* $\mathcal{R}_j \subset \{1, \ldots, N\}$. The form of this subproblem depends on whether the node is a decision node or a leaf:

- **Decision node.** The reduced subproblem becomes a weighted binary classification task with 0/1 loss, where the two classes correspond to the left and right child nodes, i.e., the only two possible outcomes. For each instance, we compute the loss of both children by propagating the instance through their respective subtrees. The instance is then assigned a pseudo-label corresponding to the child with the lower loss. Formally, the reduced objective becomes:

$$E_j(\boldsymbol{\theta}_j) = \textstyle\sum_{n \in \mathcal{R}_j} \overline{L}(\overline{y}_n, f_j(\mathbf{x}_n; \boldsymbol{\theta}_j)) + \lambda \, |\boldsymbol{\theta}_j|_1, \tag{3}$$

  where $\overline{y}_n$ is the pseudo-label and $\overline{L}$ is the corresponding loss. Although this problem is NP-hard in general, it can be effectively approximated using a convex surrogate. We use ℓ_1-regularized logistic regression, and solve it using LIBLINEAR (Fan et al., 2008).

- **Leaf node.** The reduced subproblem corresponds to minimizing the original loss function over the subset $\mathcal{R}_j$, using the leaf's GAM predictor:

$$E_j(\boldsymbol{\theta}_j) = \textstyle\sum_{n \in \mathcal{R}_j} L(\mathbf{y}_n, h_j(\mathbf{x}_n; \boldsymbol{\theta}_j)). \tag{4}$$

  When the leaf GAM is represented as an ensemble of S decision stumps, we train it using gradient boosting, a well-established and efficient framework for learning accurate ensembles with good generalization.

Using the separability property of the objective, we optimize all nodes at the same depth independently and in parallel. We adopt a reverse breadth-first traversal: starting from the deepest level (i.e., the GAM leaf nodes), we proceed upward level by level, optimizing all nodes at each depth until reaching the root. This full pass over the tree constitutes a single iteration of coordinate descent, during which all parameters are updated once. Empirically, we find that the algorithm typically converges (to a local optimum) within approximately ten such iterations.

Computational complexity of training consists of two parts: fitting the leaf GAMs and optimizing the decision nodes. For the stump-based GAMs used in our implementation, gradient boosting requires $O(S\,D\,|\mathcal{R}_j| \log |\mathcal{R}_j|)$ to fit S stumps for the set of training points $\mathcal{R}_j$ reaching leaf j, due to the need for feature sorting. Since the union of all reduced sets across leaves covers the entire training set, the total *sequential* cost of training all leaf GAMs can be loosely upper bounded by the cost of fitting S stumps via gradient boosting on the full dataset. In practice, we train all leaves in parallel, which significantly improves efficiency. Similarly, the cost of updating all decision nodes at a given depth can be upper bounded by the cost of training a single logistic regression model on the full dataset. Thus, for a tree of depth Δ, the total *sequential* cost of training all decision nodes is approximately equivalent to training Δ logistic regression models on the full training set.

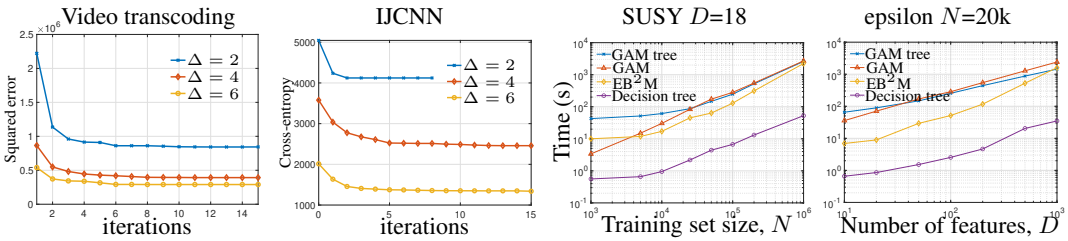


Figure 2: *Left 2 plots*: learning curves as a function optimization iterations. *Right 2 plots*: training time of GAM trees and baselines as a function of training set size N and feature dimension D. All models except GAM use parallel training with 8 threads.

5 Experiments

We begin by showing the scalability and effectiveness of the learning algorithm for GAM trees followed by the empirical comparison with existing tree-based models and standard GAM variants on regression and classification benchmarks. We then highlight the interpretability of GAM trees through two real-world case studies: used car price prediction and Airbnb rental pricing.

5.1 Training time

The left two plots in fig. 2 show the learning curves for both regression (squared error) and classification (cross entropy) datasets demonstrating the effectiveness of the alternating optimization algorithm. The right two plots show the scalability with respect to the training set size N and feature dimension D. Thanks to the parallel training of leaf GAMs on smaller reduced problems, the overall training time is similar to the sequential training of a single GAM on the whole training set, and is not much slower than training EBMs with pairwise interactions.

5.2 Performance comparison with baselines

We conduct a comparative study against several tree-based baselines, including constant-leaf trees and linear model trees. For a fair and controlled comparison, all models (except where noted) are trained using alternating optimization under the same framework. SIM Trees are included as a baseline using the original implementation provided by the authors. In addition to tree-based models, we also compare against standard flat GAM baselines: Explainable Boosting Machines (EBMs) with both univariate and pairwise interactions, widely regarded as a strong benchmark for interpretable models, and PyGAM, which implements traditional spline-based GAMs using cubic splines. All baseline models are carefully tuned by performing hyperparameter search over their important parameters. Further implementation details, more experiment results, training procedures, and dataset descriptions are provided in Appendix.

Table 1 reports results on regression benchmarks, and Table 2 shows performance on classification tasks. Overall, GAM trees consistently outperform the baseline models in terms of test error, particularly on regression datasets, while also producing shallower trees with fewer leaves. The performance advantage is especially notable on specific datasets, for instance, in the auction and video transcoding tasks, where GAM trees substantially outperform both constant- and linear-leaf trees. These improvements can be attributed to the increased modeling flexibility of GAMs in the leaves, which allows the trees to generalize well with fewer splits.

In classification tasks, constant-leaf trees perform more competitively than in regression, where piecewise-constant approximations with limited number of pieces (leaves) often struggle to model smooth relationships. Results for linear-leaf trees are omitted from classification benchmarks, as they are functionally equivalent to constant-leaf trees with an additional level of depth. On some regression benchmarks, EB^{2}M models with pairwise interactions also achieve competitive accuracy. However, these belong to a different class of models that incorporate a potentially large number of pairwise interaction terms, which can hinder interpretability. For instance, in the superconductivity dataset, EB^{2}M includes 57 pairwise terms, when each visualized as a heatmap, makes the overall model significantly more complex to interpret.

5.3 Interpretability of GAM trees

Car price prediction. To demonstrate the inherent interpretability of GAM trees with sparse oblique decision nodes, we conduct experiments using a UK car price prediction dataset available on the Kaggle platform. The prediction task involves estimating the price of used vehicles based on the

Table 1: Training and test RMSE, the depth Δ of the tree and number of leaves (average $\pm$ standard deviation over 5 runs) for different trees and GAMs. (*) In the leaves' row for EB^{2}M we specify the number of used pairwise terms. N is the dataset size, D is the feature dimension. Green color is the best test error, and blue is the second best.

Dataset		**GAM tree**	Decision tree	Linear tree	SIM tree	EB^{2}M	EBM
airfoil	train	1.16±0.04	1.76±0.07	1.83±0.09	1.85±0.13	1.35±0.08	4.36±0.05
N=1k	test	1.89±0.09	2.64±0.19	2.15±0.22	2.26±0.03	2.06±0.10	4.60±0.23
D=5	depth Δ	4.0±0.0	8.0±0.0	6.0±0.0	6.0±0.0	0±0	0±0
	leaves	16.0±0.0	120.8±14.7	36.8±2.3	17.6±0.8	5*	1±0
auction	train $\times 10^2$	6.12±0.43	11.97±1.41	15.89±0.98	10.41±0.52	24.00±0.45	51.32±0.41
N=2k	test $\times 10^2$	8.04±0.47	23.48±4.20	19.62±5.82	13.05±1.70	26.04±1.90	52.76±1.30
D=16	depth Δ	6.0±0.0	11.6±0.5	6.0±0.0	6.0±0.0	0±0	0±0
	leaves	28.8±2.0	194.2±14.1	39.0±7.2	19.4±1.0	15*	1±0
protein	train	3.61±0.05	3.74±0.08	4.35±0.04	4.03±0.02	3.92±0.01	4.67±0.01
N=46k	test	4.13±0.05	4.24±0.03	4.48±0.05	4.24±0.03	4.23±0.02	4.82±0.02
D=9	depth Δ	6.0±0.0	12.0±0.0	8.0±0.0	6.0±0.0	0±0	0±0
	leaves	60.8±2.2	1108.2±129.9	164.2±14.9	63.2±0.7	9*	1±0
supercon-	train	6.35±0.13	7.27±0.11	10.37±0.23	10.44±0.10	6.46±0.16	8.04±0.09
ductivity	test	10.29±0.13	11.39±0.10	11.94±0.16	11.96±0.22	10.30±0.13	10.90±0.06
N=21k	depth Δ	6.0±0.0	14.0±0.0	8.0±0.0	4.0±0.0	0±0	0±0
D=81	leaves	60.0±1.9	2315.4±73.7	154.2±6.1	16.0±0.0	57*	1±0
video	train	1.27±0.06	2.16±0.12	3.24±0.08	2.07±0.03	5.76±0.03	10.78±6.65
transcode	test	1.61±0.09	2.98±0.10	3.66±0.34	2.37±0.14	5.91±0.14	10.78±0.27
N=69k	depth Δ	6.0±0.0	14.0±0.0	6.0±0.0	6.0±0.0	0±0	0±0
D=24	leaves	63.4±0.8	1893.8±264.7	52.4±5.2	63.6±0.5	22*	1±0
cpuact	train	2.00±0.01	2.25±0.11	2.39±0.05	1.80±0.03	1.83±0.04	2.19±0.02
N=21k	test	2.36±0.09	2.63±0.09	2.63±0.25	2.42±0.06	2.40±0.16	2.50±0.05
D=8	depth Δ	2.0±0.0	8.0±0.0	6.4±1.0	6.0±0.0	0±0	0±0
	leaves	4.0±0.0	111.8±29.1	15.4±2.0	37.6±1.9	19*	1±0

Table 2: As in Table 1 but for binary classification datasets. The error is a 0/1 misclassification (%).

Dataset		**GAM tree**	Decision tree	SIM tree	EB^{2}M	EBM	PyGAM
ijcnn	train	0.07±0.03	0.28±0.07	1.77±0.15	1.13±0.05	4.36±0.06	4.89±0.08
N=50k	test	1.88±0.05	1.63±0.10	2.48±0.09	2.30±0.07	4.98±0.19	5.31±0.19
D=22	depth Δ	4.0±0.0	12.0±0.0	6.0±0.0	0±0	0±0	0±0
	leaves	16.0±0.0	230.6±7.9	59.2±0.7	20*	1±0	1±0
electricity	train	2.02±0.45	10.02±0.40	14.37±0.31	8.21±0.39	10.96±0.05	19.51±0.09
N=45k	test	8.20±0.23	17.55±0.45	15.37±0.29	10.59±0.20	12.32±0.39	19.80±0.31
D=14	depth Δ	6.0±0.0	14.0±0.0	6.0±0.0	0±0	0±0	0±0
	leaves	63.8±0.4	1876±76	60.2±3.8	10*	1±0	1±0
letter	train	0.65±0.21	4.16±0.47	11.51±0.21	6.21±0.10	15.93±0.19	15.78±0.08
N=20k	test	6.12±0.62	6.25±0.57	12.09±0.53	8.06±0.28	16.58±0.48	17.15±0.09
D=16	depth Δ	6.0±0.0	7.0±0.0	4.0±0.0	0±0	0±0	0±0
	leaves	59.4±1.5	71.4±5.3	16.0±0.0	15*	1±0	1±0
phoneme	train	1.98±1.21	1.99±0.53	9.73±0.55	8.06±0.46	14.99±0.40	15.90±0.15
N=5.4k	test	13.13±0.80	13.34±0.91	13.93±1.60	13.56±1.61	17.28±0.55	16.97±0.50
D=5	depth Δ	5.0±0.0	11.8±0.4	6.0±0.0	0±0	0±0	0±0
	leaves	18.8±3.2	379.6±27.9	39.8±1.7	5*	1±0	1±0
covertype	train	5.56±0.34	9.37±0.44	17.12±0.06	13.22±0.04	22.46±0.02	22.48±0.02
N=581k	test	8.47±0.27	9.98±0.38	17.22±0.07	13.97±0.14	22.68±0.12	22.72±0.10
D=54	depth Δ	8.0±0.0	12.0±0.0	6.0±0.0	0±0	0±0	0±0
	leaves	210.4±7.5	1296.8±52.1	64.0±0.0	49*	1±0	1±0

following features: make and model, year of manufacture, mileage, fuel type, transmission type, fuel efficiency (measured in miles per gallon, MPG), and annual tax. Since the dataset contains a large number of distinct car models, we limit our analysis to the 10 most frequently occurring models to avoid visual clutter from high-cardinality categorical variables. Given the diversity of vehicles in the dataset, it is reasonable to expect the presence of subgroups of similar cars that are well-suited for an additive modeling, a suitable setting for GAM trees.

The training set contains approximately 27,000 instances with 8 features, while the test set consists of 6,900 examples. After one-hot encoding of categorical variables, the feature space expands to 22 dimensions. We train a depth-3 oblique GAM Tree with a strong sparsity penalty ($\lambda = 100$) to encourage interpretability. For clarity, we constrain internal decision nodes to use only categorical

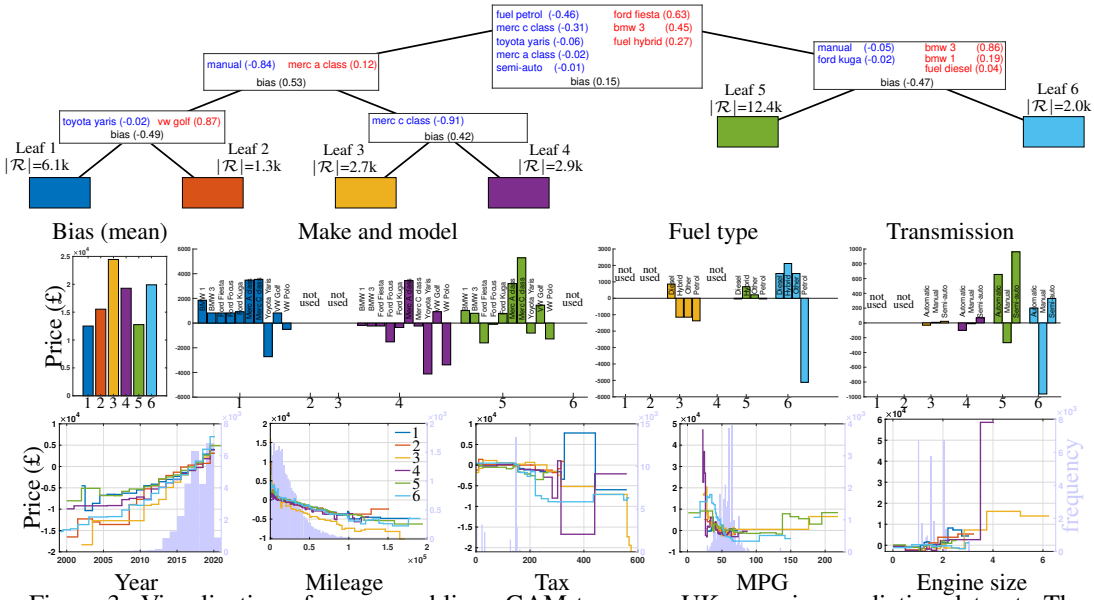


Figure 3: Visualization of a sparse oblique GAM tree on a UK car price prediction dataset. The decision nodes show the feature names of nonzero weights and the corresponding weight value (sorted from largest to smallest in magnitude). The blue (red) features contribute sending points to the left (right). The leaves are indexed from 1 to 6, and are assigned a unique color to distinguish their shape functions plotted below. $|\mathcal{R}|$ is the number of training points that reach this leaf.

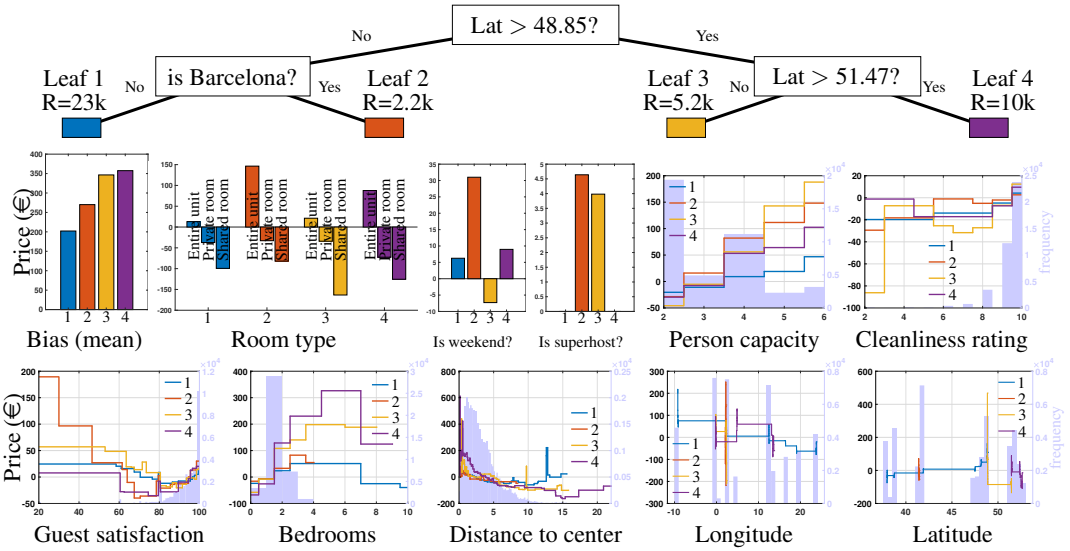


Figure 4: Visualization of a GAM tree on Airbnb price prediction dataset.

features, while allowing the leaf additive models to use the full feature set. The resulting model achieves a test RMSE of 1,631, outperforming a standard flat GAM (EBM, 2,159), a GAM with pairwise interactions (EB^{2}M, 1,773), and a traditional decision tree (CART, 1,759). Notably, the model remains highly interpretable and is visualized in fig. 3.

Decision nodes Although training began with a full depth-3 tree, the final model was pruned to a smaller structure due to the sparsity penalty. Internal decision nodes are limited to three categorical features (car model, transmission type, and fuel type) resulting in a maximum of 17 input dimensions after one-hot encoding. In practice, the learned oblique splits are significantly sparser: the most complex split, at the root, uses only 8 nonzero weights. Given the binary nature of one-hot encoding, routing through a node reduces to summing weights corresponding to active categories. This structure improves interpretability, as routing behavior can be directly linked to feature combinations. For example, all petrol vehicles that are neither Ford Fiesta nor BMW 3 Series are routed

to the left subtree at the root. BMW 3 Series vehicles consistently reach Leaf 6, while Mercedes C-Class cars are directed to Leaf 3. And many other similar patterns can be inferred by inspecting the sparse oblique split coefficients.

GAM Leaves. Each of the six leaf nodes is assigned a unique index and color, and their shape functions are plotted per feature to show variation across subpopulations. The bias term in each leaf GAM, representing the average predicted price, is shown as a bar plot. Traditional constant-valued trees can be seen as special cases of GAM trees with only the bias term active. From the bias values, we observe that Leaf 3 captures higher-priced vehicles, while Leaves 1 and 5 represent lower-price segments. We visualize shape functions for three categorical features: car model, transmission, and fuel type. Due to each leaf specializing in a narrow subset of the data, some categorical features are uninformative, for example, Leaves 2, 3, and 6 each include only one car model, and Leaves 1 and 2 only contain manual transmissions.

In the final row, shape functions for continuous features are shown alongside training data histograms (light purple, right y-axis). Sharp changes in the curves often reflect data sparsity, e.g., only 340 cars were made before 2010. In dense regions, shape functions are more consistent across leaves. Leaf 3's mileage function, for instance, declines more steeply, reflecting faster depreciation of higher-end cars like the Mercedes C-Class. This illustrates how GAM trees show contrastive patterns across different leaf subpopulations, and offer both strong predictive performance and clear local/global interpretability. For comparison, EB^2M results are shown in Appendix; its eight global interaction terms produce complex, less interpretable heatmaps.

Airbnb price prediction. While less expressive than oblique splits, axis-aligned splits offer high interpretability, especially in shallow trees. Traditional axis-aligned trees with constant-valued leaves often require many splits to reach competitive accuracy, reducing interpretability. In contrast, GAM trees achieve strong performance with far fewer splits by using more flexible leaf models. To illustrate, we apply GAM trees to Airbnb price prediction, using features such as location (city, latitude, longitude), room count, and other listing attributes (Gyodi & Nawaro, 2021). We train depth-2 and depth-3 GAM trees with axis-aligned splits restricted to geographic features (excluding the high-cardinality city feature from the leaf GAMs for clarity). Both models outperform a flat GAM (RMSE 112.6) and EB^2M (RMSE 99.2), achieving test RMSEs of 98.2 (depth 2) and 95.3 (depth 3). A standard decision tree with constant leaves achieves a slightly better RMSE (93.9) but uses over 7,000 leaves, making it uninterpretable. Fig. 4 shows the depth-2 tree; results for the depth-3 are in Appendix.

Decision nodes. All splits are based on location, producing intuitive geographic groupings. For example, listings from Barcelona are routed to Leaf 2, while other southern cities (latitude < 48.85) go to Leaf 1. Northern cities are split between Leaf 3 (mainly Paris) and Leaf 4 (Amsterdam and Berlin), with London divided across both based on a latitude threshold of 51.47.

GAM leaves. Leaf bias terms reveal that Leaf 1 (southern cities) contains lower-priced listings than Leaves 3 and 4. Room type effects are generally consistent across leaves, though price differences are smallest in Leaf 1. The weekend effect is modest overall but stronger in Barcelona (Leaf 2), where it adds €30. The superhost badge has minimal impact, unused entirely by Leaves 1 and 4. For numerical features, shape functions show clear trends: prices increase with capacity (bedrooms, occupancy), and proximity to the city center correlates with higher prices. Geographic features (longitude, latitude) exhibit region-specific gradients, as each leaf GAM is localized. Overall, this example shows how GAM trees combine simple, interpretable decision structures with flexible, localized modeling to achieve both accuracy and transparency.

6 Conclusion

We have proposed a natural generalization of GAMs and decision trees, the GAM tree, and a training algorithm based on tree alternating optimization to optimize all parameters jointly in a scalable way. The GAM tree remains a highly interpretable model but it can improve accuracy over both GAMs and trees by a wide margin. We expect GAM trees to be particularly useful with tabular data in domains such as medicine, customer relationship management, finance or law, where interpretability is highly valuable. In such domains one typically finds natural populations (e.g. on the basis of gender, geography, etc.) for which the target variable may well be modeled additively. We are currently working on combining GAM trees with sparsity of the leaf GAMs so as to achieve local feature selection.

AI USE STATEMENT

In this work, we did not use generative AI tools.

ETHICS STATEMENT

This work proposes an inherently interpretable model whose decisions can be inspected directly, which we view as beneficial in high-stakes domains. All experiments use publicly available datasets, which we cite together with their sources. We are not aware of ethical concerns specific to this work.

REPRODUCIBILITY STATEMENT

The GAM tree model and its training algorithm are defined in sections 3 and 4, with pseudocode in fig. 5 (appendix). Appendix C lists the datasets and their sources, the implementation details of our method (including library versions), the hardware used, and the hyperparameter grids searched for our method and for every baseline. All comparison results are averaged over 5 runs with different train/test splits and reported with standard deviations.

REFERENCES

Rishabh Agarwal, Levi Melnick, Nicholas Frosst, Xuezhou Zhang, Ben Lengerich, Rich Caruana, and Geoffrey E. Hinton. Neural additive models: Interpretable machine learning with neural nets. In M. Ranzato, A. Beygelzimer, P.S. Liang, J. W. Vaughan, and Y. Dauphin (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34. MIT Press, Cambridge, MA, 2021.

Sercan Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35:6679–6687, 05 2021. doi: 10.1609/aaai.v35i8.16826.

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

Leo J. Breiman, Jerome H. Friedman, R. A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, Calif., 1984.

Miguel Á. Carreira-Perpiñán and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pp. 1211–1221. MIT Press, Cambridge, MA, 2018.

Kin-Yee Chan and Wei-Yin Loh. LOTUS: An algorithm for building accurate and comprehensible logistic regression trees. *Journal of Computational and Graphical Statistics*, 13(4):826–852, 2004.

Chun-Hao Chang, Sarah Tan, Ben Lengerich, Anna Goldenberg, and Rich Caruana. How interpretable and trustworthy are GAMs? In *Proc. of the 27th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2021)*, pp. 95–105, Online, August 14–18 2021.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proc. of the 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2016)*, pp. 785–794, San Francisco, CA, August 13–17 2016.

Emir Demirović, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Peter J. Stuckey. MurTree: Optimal decision trees via dynamic programming and search. *J. Machine Learning Research*, 23(26):1–47, 2022.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *J. Machine Learning Research*, 9:1871–1874, August 2008.

Sebastian Fischer, Liana Harutyunyan, Matthias Feurer, and Bernd Bischl. Openml-ctr23 – a curated tabular regression benchmarking suite. AutoML 2023 Workshop Track, 2023.

Kristof Gyodi and Lukasz Nawaro. Determinants of airbnb prices in european cities: A spatial econometrics approach. *Tourism Management*, 86:104319, 2021.

Trevor J. Hastie and Robert J. Tibshirani. *Generalized Additive Models*. Number 43 in Monographs on Statistics and Applied Probability. Chapman & Hall, London, New York, 1990.

Trevor J. Hastie, Robert J. Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning—Data Mining, Inference and Prediction*. Springer Series in Statistics. Springer-Verlag, second edition, 2009.

Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, May 1976.

Rasul Kairgeldin and Miguel Á. Carreira-Perpiñán. Bivariate decision trees: Smaller, interpretable, more accurate. In *Proc. of the 30th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2024)*, pp. 1336–1347, Barcelona, Spain, August 25–29 2024.

Ron Kohavi. Scaling up the accuracy of naive-Bayes classifiers: A decision-tree hybrid. In Evangelos Simoudis, Jiawei Han, and Usama Fayyad (eds.), *Proc. of the 2nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 1996)*, pp. 202–207, Portland, OR, August 2–4 1996.

Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. *Machine Learning*, 95(1–2): 161–205, 2005.

M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013.

Jiachang Liu, Chudi Zhong, Margo Seltzer, and Cynthia Rudin. Fast sparse classification for generalized linear and additive models. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera (eds.), *Proc. of the 25th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2022)*, pp. 9304–9333, Online, March 28–30 2022.

Wei-Yin Loh. Fifty years of classification and regression trees. *International Statistical Review*, 82 (3):329–348 (with discussion, pp. 349–370), December 2014.

Yin Lou, Rich Caruana, and Johannes Gehrke. Intelligible models for classification and regression. In *Proc. of the 18th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2012)*, pp. 150–158, August 21–24 2012.

Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. In *Proc. of the 19th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2013)*, pp. 623–631, Chicago, IL, August 11–14 2013.

David Lubinsky. Classification trees with bivariate splits. *Applied Intelligence*, 4(3):283–296, July 1994.

Giampiero Marra and Simon N. Wood. Practical variable selection for generalized additive models. *Computational Statistics and Data Analysis*, 55(7):2372–2387, July 2011.

J. R. Quinlan. Learning with continuous classes. In Anthony Adams and Leon Sterling (eds.), *Proc. 5th Australian Joint Conf. on Artificial Intelligence (AI'92)*, pp. 343–348, 1992.

J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

Pradeep Ravikumar, John Lafferty, Han Liu, and Larry Wasserman. Sparse additive models. *J. Statistical Society B*, 71(5):1009–1030, November 2009.

Lior Rokach and Oded Maimon. *Data Mining With Decision Trees. Theory and Applications*. Number 81 in Series in Machine Perception and Artificial Intelligence. World Scientific, second edition, 2015.

Agus Sudjianto, Zebin Yang, and Aijun Zhang. Single-index model tree. *IEEE Trans. Knowledge and Data Engineering*, 35(3):3101–3114, March 2023.

Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, fourth edition, 2016.

Simon N. Wood. *Generalized Additive Models: An Introduction with R*. Texts in Statistical Science. Chapman & Hall/CRC, second edition, 2017.

Achim Zeileis, Torsten Hothorn, and Kurt Hornik. Model-based recursive partitioning. *Journal of Computational and Graphical Statistics*, 17(2):492–514, 2008.

Arman Zharmagambetov and Miguel Á. Carreira-Perpiñán. Smaller, more accurate regression forests using tree alternating optimization. In Hal Daumé III and Aarti Singh (eds.), *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, pp. 11398–11408, Online, July 13–18 2020.

APPENDIX

We provide the pseudocode of the learning algorithm for GAM trees in fig. 5, more complete comparison results in Tables 3 and 4, comparison with additional baselines in Tables 5 and 6, the visualization of EB^{2}M with pairwise interactions for the car price prediction dataset in fig. 6, the visualization of the depth-3 axis-aligned GAM tree for the Airbnb dataset in fig. 7 and finally, the description of the experiments' setup for reproducibility: datasets, comparison methods, and hyperparameters in section C.

A LIMITATIONS

Local optima. The TAO algorithm is a form of coordinate descent, which optimizes one node at a time. It converges to a local optimum that depends on the initialization. We initialize with random weights and find that results are generally stable across runs (as shown in our experiments by the small standard deviations), but global optimality is not guaranteed.

Modeling assumption. Each leaf GAM assumes that the prediction decomposes as a sum of univariate functions, i.e., it cannot model feature interactions within a leaf. While the tree structure captures some interactions by routing instances to different leaves, interactions that exist within a single leaf population are not modeled.

Sparsity in leaf GAMs. While the decision nodes can be made sparse via the ℓ_1 penalty, we do not enforce sparsity over the leaf GAMs, meaning each leaf GAM uses all D features. Proper feature selection within leaf GAMs would further improve interpretability but appears not to have been well studied and is left for future work.

Multiclass classification. In this work we focus on binary classification (using a sigmoid link function) and regression. Extension to multiclass problems (e.g., using a softmax) is straightforward in principle but was not investigated. We should note that even traditional GAMs have been rarely used for multiclass problems.

B PSEUDOCODE

input initial GAM tree $T(\cdot, \boldsymbol{\Theta})$ of depth Δ,
training set $\{\mathbf{x}_n, y_n\}_{n=1}^N$, number of stumps S,
regularization parameter $\lambda \geq 0$.
repeat
  for $d = \Delta$ **to** 0 **do**
    for all nodes $node$ at depth d
      if $node$ is a leaf
        fit S decision stumps using gradient boosting
        using squared error for regression
        or cross entropy for classification
      else
        fit a weighted ℓ_1-regularized logistic regression
        with parameter λ
      end if
    end for
  end for
until stopping criteria
return trained GAM tree $T(\cdot; \boldsymbol{\Theta})$

Figure 5: Pseudocode of the alternating optimization algorithm for learning GAM trees.

Table 3: Training and test RMSE, the depth Δ of the tree and number of leaves (average $\pm$ standard deviation over 5 runs) for different trees and GAMs. (*) In the leaves' row for EB^{2}M we specify the number of used pairwise terms. N is the dataset size, D is the feature dimension. Green color is the best test error, and blue is the second best.

Dataset		**GAM tree**	Decision tree	Linear tree	SIM tree	EB^{2}M	EBM	PyGAM
airfoil	train	1.16±0.04	1.76±0.07	1.83±0.09	1.85±0.13	1.35±0.08	4.36±0.05	4.38±0.06
N=1k	test	1.89±0.09	2.64±0.19	2.15±0.22	2.26±0.03	2.06±0.10	4.60±0.23	4.59±0.22
D=5	depth Δ	4.0±0.0	8.0±0.0	6.0±0.0	6.0±0.0	0±0	0±0	0±0
	leaves	16.0±0.0	120.8±14.7	36.8±2.3	17.6±0.8	5*	1±0	1±0
auction	train $\times 10^2$	6.12±0.43	11.97±1.41	15.89±0.98	10.41±0.52	24.00±0.45	51.32±0.41	51.18±0.38
N=2k	test $\times 10^2$	8.04±0.47	23.48±4.20	19.62±5.82	13.05±1.70	26.04±1.90	52.76±1.30	52.85±1.46
D=16	depth Δ	6.0±0.0	11.6±0.5	6.0±0.0	6.0±0.0	0±0	0±0	0±0
	leaves	28.8±2.0	194.2±14.1	39.0±7.2	19.4±1.0	15*	1±0	1±0
protein	train	3.61±0.05	3.74±0.08	4.35±0.04	4.03±0.02	3.92±0.01	4.67±0.01	4.78±0.01
N=46k	test	4.13±0.05	4.24±0.03	4.48±0.05	4.24±0.03	4.23±0.02	4.82±0.02	4.85±0.03
D=9	depth Δ	6.0±0.0	12.0±0.0	8.0±0.0	6.0±0.0	0±0	0±0	0±0
	leaves	60.8±2.2	1108.2±129.9	164.2±14.9	63.2±0.7	9*	1±0	1±0
supercon-	train	6.35±0.13	7.27±0.11	10.37±0.23	10.44±0.10	6.46±0.16	8.04±0.09	10.32±0.02
ductivity	test	10.29±0.13	11.39±0.10	11.94±0.16	11.96±0.22	10.30±0.13	10.90±0.06	12.30±0.20
N=21k	depth Δ	6.0±0.0	14.0±0.0	8.0±0.0	4.0±0.0	0±0	0±0	0±0
D=81	leaves	60.0±1.9	2315.4±73.7	154.2±6.1	16.0±0.0	57*	1±0	1±0
video	train	1.27±0.06	2.16±0.12	3.24±0.08	2.07±0.03	5.76±0.03	10.78±6.65	10.80±0.07
transcode	test	1.61±0.09	2.98±0.10	3.66±0.34	2.37±0.14	5.91±0.14	10.78±0.27	10.79±0.27
N=69k	depth Δ	6.0±0.0	14.0±0.0	6.0±0.0	6.0±0.0	0±0	0±0	0±0
D=24	leaves	63.4±0.8	1893.8±264.7	52.4±5.2	63.6±0.5	22*	1±0	1±0
sarcos	train	2.50±0.07	2.94±0.10	2.90±0.04	2.86±0.03	2.81±0.06	4.16±0.02	4.43±0.01
N=49k	test	3.30±0.04	3.78±0.09	3.08±0.06	3.14±0.04	3.48±0.02	4.68±0.05	4.64±0.05
D=21	depth Δ	6.0±0.0	12.0±0.0	6.0±0.0	6.0±0.0	0±0	0±0	0±0
	leaves	42.0±1.7	1271.0±105.1	59.4±2.8	63.6±0.5	19*	1±0	1±0
cpuact	train	2.00±0.01	2.25±0.11	2.39±0.05	1.80±0.03	1.83±0.04	2.19±0.02	2.53±0.02
N=21k	test	2.36±0.09	2.63±0.09	2.63±0.25	2.42±0.06	2.40±0.16	2.50±0.05	2.69±0.06
D=8	depth Δ	2.0±0.0	8.0±0.0	6.4±1.0	6.0±0.0	0±0	0±0	0±0
	leaves	4.0±0.0	111.8±29.1	15.4±2.0	37.6±1.9	19*	1±0	1±0

Table 4: As in Table 1 but for binary classification datasets. The error is a 0/1 misclassification (%).

Dataset		**GAM tree**	Decision tree	SIM tree	EB^{2}M	EBM	PyGAM
ijcnn	train	0.07±0.03	0.28±0.07	1.77±0.15	1.13±0.05	4.36±0.06	4.89±0.08
N=50k	test	1.88±0.05	1.63±0.10	2.48±0.09	2.30±0.07	4.98±0.19	5.31±0.19
D=22	depth Δ	4.0±0.0	12.0±0.0	6.0±0.0	0±0	0±0	0±0
	leaves	16.0±0.0	230.6±7.9	59.2±0.7	20*	1±0	1±0
electricity	train	2.02±0.45	10.02±0.40	14.37±0.31	8.21±0.39	10.96±0.05	19.51±0.09
N=45k	test	8.20±0.23	17.55±0.45	15.37±0.29	10.59±0.20	12.32±0.39	19.80±0.31
D=14	depth Δ	6.0±0.0	14.0±0.0	6.0±0.0	0±0	0±0	0±0
	leaves	63.8±0.4	1876±76	60.2±3.8	10*	1±0	1±0
letter	train	0.65±0.21	4.16±0.47	11.51±0.21	6.21±0.10	15.93±0.19	15.78±0.08
N=20k	test	6.12±0.62	6.25±0.57	12.09±0.53	8.06±0.28	16.58±0.48	17.15±0.09
D=16	depth Δ	6.0±0.0	7.0±0.0	4.0±0.0	0±0	0±0	0±0
	leaves	59.4±1.5	71.4±5.3	16.0±0.0	15*	1±0	1±0
phoneme	train	1.98±1.21	1.99±0.53	9.73±0.55	8.06±0.46	14.99±0.40	15.90±0.15
N=5.4k	test	13.13±0.80	13.34±0.91	13.93±1.60	13.56±1.61	17.28±0.55	16.97±0.50
D=5	depth Δ	5.0±0.0	11.8±0.4	6.0±0.0	0±0	0±0	0±0
	leaves	18.8±3.2	379.6±27.9	39.8±1.7	5*	1±0	1±0
phishing	train	2.97±0.30	1.61±0.11	6.06±0.10	5.55±0.21	5.91±0.08	5.86±0.05
websites	test	4.41±0.36	3.74±0.30	6.23±0.30	5.68±0.21	5.86±0.17	5.91±0.25
N=11k	depth Δ	4.0±0.0	10.4±0.5	2.0±0.0	0±0	0±0	0±0
D=68	leaves	14.4±1.2	132.8±7.0	4.0±0.0	14*	1±0	1±0
covertype	train	5.56±0.34	9.37±0.44	17.12±0.06	13.22±0.04	22.46±0.02	22.48±0.02
N=581k	test	8.47±0.27	9.98±0.38	17.22±0.07	13.97±0.14	22.68±0.12	22.72±0.10
D=54	depth Δ	8.0±0.0	12.0±0.0	6.0±0.0	0±0	0±0	0±0
	leaves	210.4±7.5	1296.8±52.1	64.0±0.0	49*	1±0	1±0

Table 5: Comparison with additional baselines: TabNet (Arik & Pfister, 2021), Neural Additive Model (NAM; Agarwal et al., 2021) and XGBoost (Chen & Guestrin, 2016). For NAM we perform grid-search on the following hyperparameter values: dropout $\{0.0, 0.1, 0.3\}$, weight decay $\{10^{-6}, 10^{-5}, 10^{-4}\}$, and a learning rate $\{0.001, 0.01\}$. For TabNet, we explore the following hyperparameters: `n_d` $\{8, 16, 64\}$, `n_steps` $\{3, 5, 7\}$, and as suggested in the documentation, we set `n_a` = `n_d`. For XGBoost, we tune these hyperparameters: `depth` $\{4, 6, 8\}$ and `n_estimators` $\{100, 500, 1000\}$. We report regression RMSE results (mean $\pm$ std).

Dataset		GAM tree	TabNet	NAM	XGBoost
cpuact	train	2.00 ± 0.01	2.45 ± 0.07	3.38 ± 0.26	0.55 ± 0.00
	test	2.36 ± 0.09	2.52 ± 0.07	3.41 ± 0.28	2.22 ± 0.05
airfoil	train	1.16 ± 0.04	3.79 ± 0.34	4.83 ± 0.47	0.05 ± 0.00
	test	1.89 ± 0.09	3.99 ± 0.43	4.87 ± 0.53	1.33 ± 0.09
auction	train	6.12 ± 0.43	36.78 ± 2.48	75.07 ± 2.17	0.41 ± 0.02
	test	8.04 ± 0.47	35.49 ± 2.92	75.30 ± 3.01	4.18 ± 0.69
protein	train	3.61 ± 0.05	3.11 ± 0.11	5.13 ± 0.24	0.81 ± 0.01
	test	4.13 ± 0.05	3.73 ± 0.11	5.15 ± 0.19	3.50 ± 0.02
superconductivity	train	6.35 ± 0.13	10.49 ± 0.35	15.82 ± 0.27	5.13 ± 0.05
	test	10.29 ± 0.13	11.46 ± 0.31	15.88 ± 0.22	9.30 ± 0.21

Table 6: As Table 5 but for classification datasets. We report 0/1 misclassification error (%, mean $\pm$ std)

Dataset		GAM tree	TabNet	NAM	XGBoost
ijcnn	train	0.07 ± 0.03	0.29 ± 0.03	7.51 ± 0.44	0.00 ± 0.00
	test	1.88 ± 0.05	1.01 ± 0.14	7.48 ± 0.55	1.20 ± 0.07
electricity	train	2.02 ± 0.45	15.44 ± 0.62	18.20 ± 0.37	0.06 ± 0.01
	test	8.20 ± 0.23	18.59 ± 0.71	18.55 ± 0.43	6.26 ± 0.15
letter	train	0.65 ± 0.21	0.24 ± 0.06	21.54 ± 1.10	0.00 ± 0.00
	test	6.12 ± 0.62	1.79 ± 0.26	22.53 ± 1.88	2.14 ± 0.22
phoneme	train	1.98 ± 1.21	10.97 ± 1.11	16.92 ± 1.34	0.00 ± 0.00
	test	13.13 ± 0.80	14.17 ± 0.54	17.55 ± 0.88	10.21 ± 1.02

C EXPERIMENTS SETUP

For all methods, including ours, we tune key hyperparameters using a holdout set. Once the best configuration is found, we run five experiments using different train/test splits and report the mean and standard deviation.

Implementation of GAM trees We implement GAM trees with the combination of Python and C++. The decision node optimization happens in C++. We use LIBLINEAR version 2.43. The leaf GAM fitting is done in Python with scikit-learn gradient boosting implementation with decision stumps by setting `max_depth` = 1 and `learning_rate` = 0.1. We perform grid search on the following hyperparameter values: number of stumps S= {500, 1000, 2000}, maximum depth $\Delta = \{2, 4, 6, 8\}$. We do not tune the decision node sparsity penalty λ, and use the default value $\lambda = 0.1$. As an initial GAM tree, we use a complete tree of depth Δ with random initial decision node parameters.

Hardware. All experiments are performed on a CPU Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz with 128 GB RAM.

The specific details of the baselines and tuned hyperparameters:

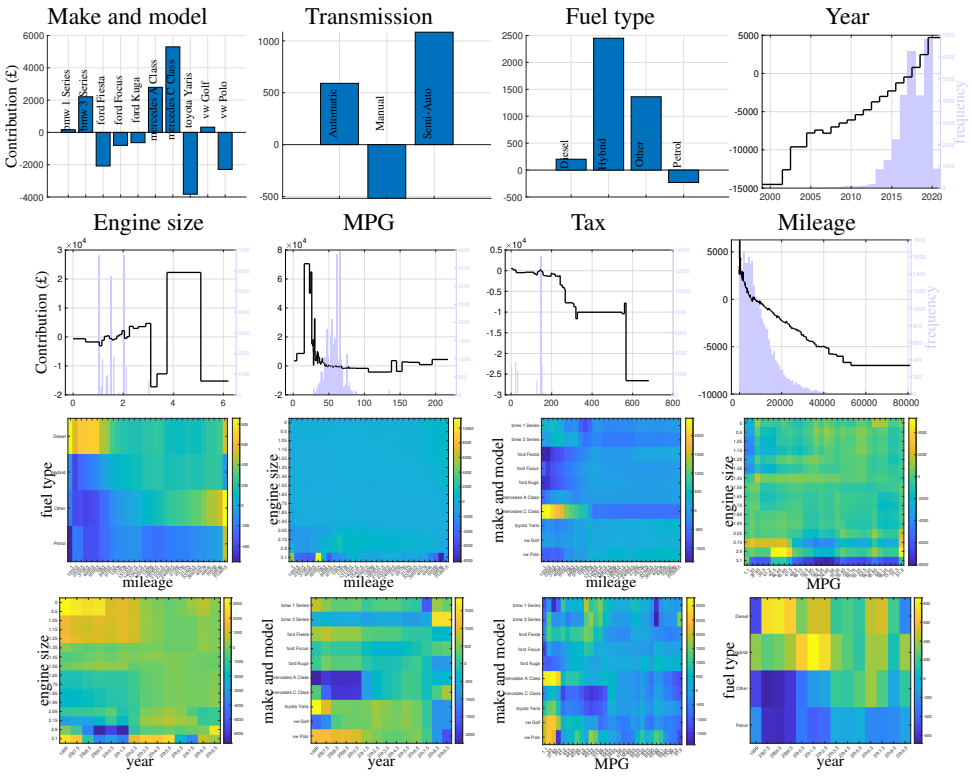


Figure 6: Visualization of EB^{2}M with pairwise interactions for car price prediction dataset. The bias (mean) is £15,200. The model has $E_{\text{train}} = 1579$ and $E_{\text{test}} = 1773$ root-mean-squared-error.

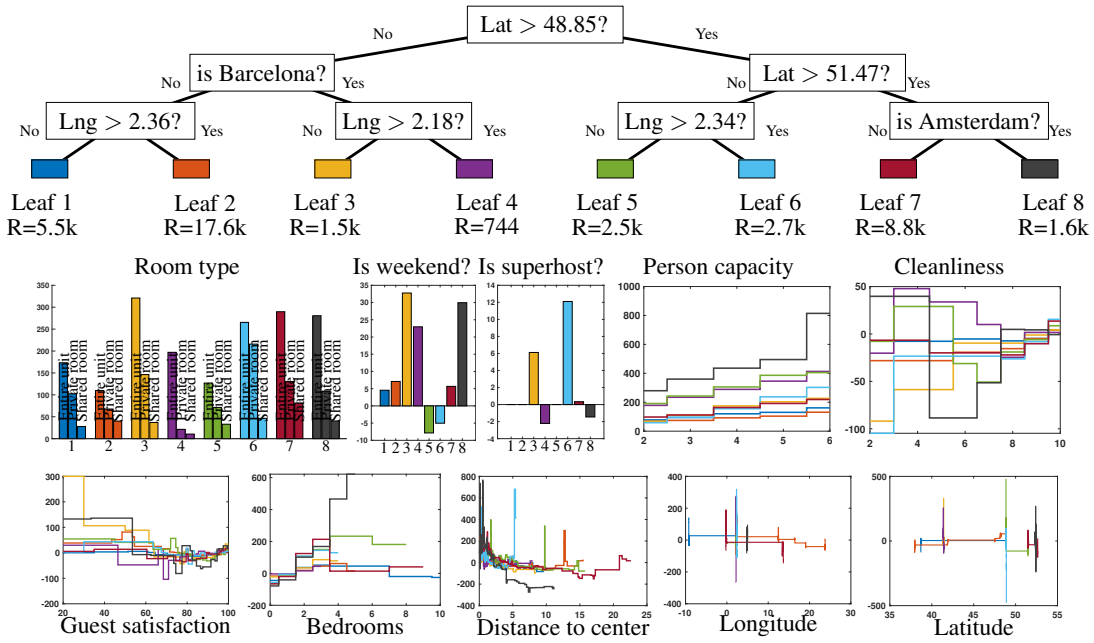


Figure 7: Visualization of a depth-3 axis-aligned GAM tree on Airbnb price prediction dataset.

Linear tree: We use as a basis our own GAM tree implementation, and change the leaf model to scikit-learn's Ridge regression. In addition to the tree depth Δ, we tune its `alpha` hyperparameter with the following values: {0.1, 1.0, 10.0}.

Constant tree: Similarly, we use as a basis our own GAM tree implementation, and change the leaf model to constant prediction. We tune the tree depth with following values $\Delta \in \{4, 6, 8, 10, 12, 14\}$.

Explainable Boosting Machine (EBM and EB^{2}M): we use the official implementation from the `interpret` Python library[4]. We perform grid search on the following hyperparameter values: `learning_rate`: {0.005, 0.01, 0.05}, `max_bins`: {512, 1024, 2048}, `min_samples_leaf`: {2, 4, 8}. We set the `interactions` parameter to 0 to use only univariate terms for EBM. `max_rounds` is set to 25 000 with `early_stopping_rounds` 50. For pairwise interactions EB^{2}M, we additionally tune `interactions`: {0.2, 0.5, 0.7, 0.9}. The version of the `interpret` package is 0.6.1.

PyGam: this implements traditional cubic splines. We use the PyGam package[5] in Python. We perform grid search on the following hyperparameters: strength of the smoothing penalty $\lambda = \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$, and the number of knots $= \{10, 20, 50, 100\}$. The version of the PyGam package is 0.9.1.

Single Index Model (SIM) tree: we use the original implementation of Sudjianto et al. (2023)[6]. We perform grid search on the following hyperparameters: `reg_lambda` = {0.0, 0.1, 0.01}, `reg_gamma` = {10^{-3}, 10^{-5}}, `max_depth` = {2, 4, 6, 8}.

Regression datasets. All of the used regression datasets come from the "OpenML-CTR23 A curated tabular regression benchmarking suite" (Fischer et al., 2023).

Classification datasets description:

IJCNN The dataset comes from an IJCNN 2001 competition. We obtain it from the LIBSVM binary dataset collection: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

electricity The goal is to identify whether the electricity price will go up or down based on the features such as demand price and day of the week. Obtained from OpenML with task ID 213.

Letter English letter recognition task, available in the UCI ML Repository (Lichman, 2013). We convert it into binary classification by combining the letters A-M into one class and the rest to the other class.

Phoneme The task is to classify between nasal and oral sounds based on phoneme descriptors. Obtained from OpenML with task ID 9952.

Phishing websites The goal is find out whether the website is fraudulent based on the various features of URL. Obtained from OpenML with task ID 14952.

Covtype Classification task of pixels into 7 forest cover types based on attributes such as elevation, aspect, soil-type, slope, etc. We turn it into binary classification by as the majority class vs the rest. Sourced from the UCI ML Repository (Lichman, 2013).

[4] https://github.com/interpretml/interpret
[5] https://github.com/dswah/pyGAM
[6] https://github.com/ZebinYang/SIMTree