Multistage Defer Trees for Hybrid Interpretability: If at First You Can't Succeed, Tree Again

Zakk Heile*
Department of Computer Science
Duke University
Durham, USA
zakk.heile@duke.edu

Hayden McTavish*
Department of Computer Science
Duke University
Durham, USA
hayden.mctavish@duke.edu

Margo Seltzer
Department of Computer Science
University of British Columbia
Vancouver, Canada
mseltzer@cs.ubc.ca

Cynthia Rudin
Department of Computer Science
Duke University
Durham, USA
cynthia@cs.duke.edu

Abstract

Recent work has shown that well-optimized individual decision trees can match complex black box models in some settings, primarily in noisy domains. For the remaining settings, however, complex ensembled compositions of trees often achieve higher accuracy at the cost of interpretability, leaving practitioners with difficult modeling decisions along an accuracy-interpretability tradeoff. Ideally, we would like to classify as much of the data as possible with one or a small number of trees, achieving interpretability for most samples while maintaining state-of-the-art accuracy. We introduce Multistage Defer Trees: a sequence of sparse decision trees that each make predictions for most samples, while deferring a small proportion to the next tree in the sequence or, ultimately, to a black box. We demonstrate that we can train this model class to match the performance of complex tree-based ensembles while routing most samples through only one or a small number of sparse decision trees. We discuss a range of techniques for training these models while maintaining simplicity. Our method expands the accuracy–interpretability frontier in settings where single-tree methods remain insufficient, demonstrating that even when complex models are necessary, they need not be fully opaque.

Code: https://github.com/zakk-h/MDT/

1 Introduction

A central challenge in interpretable machine learning is to construct interpretable models that match state-of-the-art predictive performance. Well-optimized simple models, such as optimal sparse trees, can achieve this goal in certain settings, such as when the data is generated by noisy processes [11, 70, 71]. In such cases, the accuracy-interpretability gap is not observed. Still, there remains such a gap in datasets falling outside of this regime.

One proposed direction to navigate the accuracy-interpretability gap is to use a hybrid- or partially-interpretable model [26, 30, 40, 62, 76, 77]. The goal is to learn a simple interpretable model that, for each sample, either makes a prediction for that sample or defers to a black box's prediction. By

*Equal contribution. Corresponding authors: zakk.heile@duke.edu and hayden.mctavish@duke.edu.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

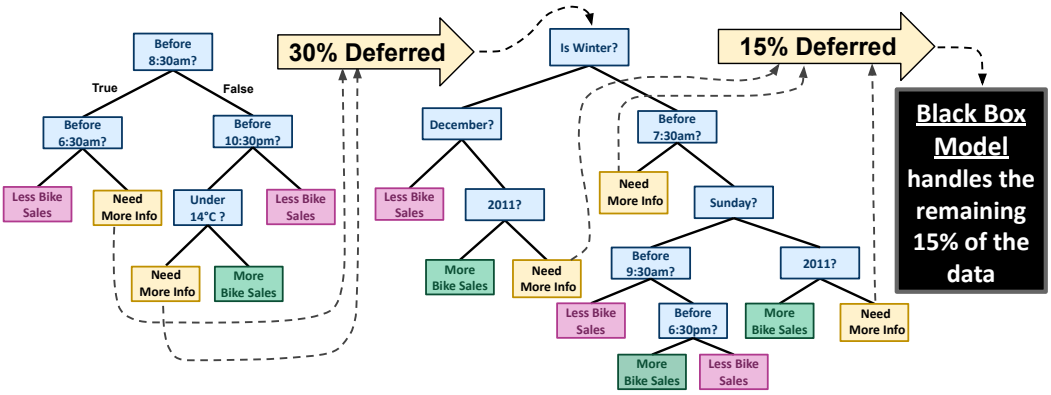


Figure 1: Example of an MDT on the Bike dataset. The model consists of two stages, each with bounded depth. Two leaves in the first stage defer to the second stage, and three leaves in the second stage defer to a black box (here, an XGBoost model). The interpretable component has 13 leaves.

specifying the proportion of samples classified by a black box (the deferral rate), users can navigate a tradeoff between matching black box accuracy and keeping decisions interpretable for more samples. While hybrid-interpretable models provide a smooth way to navigate compromises between accuracy and interpretability, existing methods often struggle to maintain black box accuracy without deferring the vast majority of samples.

We present a tree-based hybrid-interpretable model that minimizes deferral rate while matching the performance of more complex ensembles. To learn this model, we iteratively focus on subsets of the data where simple models struggle to perform as well as a black box model. This leads to a new model class, Multistage Defer Trees (MDTs), where each tree can either make a prediction or defer to the next stage; the final stage may then defer some samples to a black box model if required.

To train MDTs, we introduce an alternating optimization procedure in which the deferred-to model evolves with each stage of the MDT. As this process unfolds, the set of deferred points decreases monotonically, and the later stages and fallback model become increasingly specialized to those points. This allows the model to defer only on a small fraction of inputs while maintaining high accuracy. When the fallback is a sparse model (i.e., a single decision tree or EBM [42]), even these deferral cases remain interpretable; when the fallback is a black box, performance can be matched with minimal reliance on that model. Crucially, because deferred points are confined to a tightly constrained region of the feature space, the fallback model can be compressed, producing simpler but equivalent representations of even complex ensembles.

Our contributions are as follows.

- We introduce Multistage Defer Trees (MDTs), a new model class that allocates complexity adaptively, based on the difficulty of different regions of the feature space.
- We develop an iterative algorithm for training MDTs that successively narrows the deferral region, while borrowing strength from surrounding regions, to learn specialized trees that generalize well.
- We provide algorithms to compress MDTs into sparse single-tree or sparser sequential representations and extend this framework to compress the fallback model when it is a tree or tree ensemble.
- We demonstrate improved accuracy–deferral–sparsity trade-offs where such trade-offs exist.

2 Related Work

Hybrid Approaches Wang [76] introduces hybrid models for interpretability: an interpretable rule set or linear model makes predictions on some of the data, while deferring to fixed black box predictions on the remaining data. This allows users to navigate a tradeoff between the model accuracy

and the proportion of samples that have interpretable model predictions. This framework, in turn, connects to work in deferral models [43, 53], rejection learning [17, 68], selective classification [22], and abstention-based learning [29, 69]. Hybrid models have primarily been applied to simple rule set models or linear models [26, 30, 62, 76, 77]. One recent work explores a gradient-based method to learn a complex (100-1000 leaves) decision tree, which defers to a black box based on computing the proximity to the decision boundary [40]; by contrast, our work learns a sparser sequence of trees that directly defer where the black box is most helpful, and iteratively reduces this deferral region.

Decision Tree Optimization Decision trees are classically optimized with highly scalable greedy algorithms [14, 64]. Such algorithms can be ideal for popular ensembles of many trees [12, 16]. More recent work, however, revisits the use of less scalable, higher quality individual tree algorithms, to reduce the need for this level of model complexity and to allow use of a single well-performing sparse tree classifier for domains where interpretability is needed. In recent years, global optimization of individual trees has become reasonably tractable for bounded depth and sparsity [3, 8, 15, 20, 35, 41, 47, 74]. This problem is nevertheless NP-hard and existing solutions struggle to scale to cases with many features. Recent work sacrifices global optimality in exchange for improved scalability and runtime while maintaining performance empirically close to optimal [5, 9, 37]. Other approaches add anytime behavior, allowing early termination with high-quality solutions [21, 36, 38].

We fit a small sequence of trees that nearly matches the performance of a significantly more complex ensemble. Because MDTs rely on only a few sparse trees, we can afford to optimize each component tree more carefully than in standard greedy tree induction while still requiring the overall procedure to scale. Given this need for both higher-quality component trees and scalability, we use a version of the polynomial-time LicketySPLIT algorithm from Babbar et al. [5], which we modify to support sample weights and deferral; our theory provides guarantees achievable when each component tree is instead exactly optimal with respect to misclassification error, deferral rate, and tree complexity. Many less-greedy decision tree algorithms, including LicketySPLIT, operate on binarized features, which are often constructed using threshold guessing [48]; we use the same binarization method.

Ensemble-Guided Tree Search Vidal and Schiffer [75] present an algorithm to find the sparsest single decision tree with 0-1 predictive behavior identical to that of a given tree-based ensemble (though not necessarily identical predicted probabilities), inspired by an approach approximating Random Forest behavior with a single tree [13]. They additionally provide practical relaxations to allow for faster computation (removing the certificate that the tree is the sparsest equivalent form) or sparser representations (providing extra pruning in areas with no support in the training data). Sagi and Rokach [65] translate an XGBoost tree ensemble [16] to a single tree, with some approximation loss. McTavish et al. [48] take a slightly different approach to learn a single tree when provided with a tree-based ensemble, using the ensemble's predictions to provide heuristic pruning of the search space for an otherwise globally optimal tree. All these methods afford more interpretable single tree alternatives when provided with an ensemble model, though to guarantee matching performance with that ensemble, these approaches often find trees with many more leaves than is ideal for interpretability.

In the hybrid component of our own algorithm, we incorporate information from a black box model in a fundamentally different way than the above approaches do. Our goal is not to match the black box model's predictions exactly, nor to accelerate an optimal tree search. Instead our goal is to use the black box to resolve subspaces of the data for which interpretable algorithms cannot find an accurate simple model. This allows us to find models that are transparent on much of the data, without sacrificing meaningful accuracy relative to the black box. While we discuss several useful properties when the black box is a tree-based ensemble, our approach inherits the more general-purpose properties of hybrid models and can work with any black box model, since we only need to be able to train the black box and obtain its predictions.

3 Methodology

Notation. Let $\mathcal{X} \subseteq \mathbb{R}^p$ denote an input space with p features and for simplicity, let $\mathcal{Y} = \{0, 1\}$ denote the label space for binary classification (we additionally support multiclass classification and regression in our code). Training data is denoted as $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N \subseteq \mathcal{X} \times \mathcal{Y}$. When this data is weighted, we denote the dataset as $\mathcal{D}^w$ for weight vector $w \in \mathbb{R}^N_{\geq 0}$. We write $\mathbb{1}\{\cdot\}$ for the indicator

function. For a binary decision tree T, let $|T|$ denote its number of leaves, and let $T(x)$ denote the label predicted by T for input x.

A standard binary decision tree recursively partitions $\mathcal{X}$ by internal boolean split nodes and assigns a prediction at each leaf. We generalize this object by allowing a leaf to output either a class label or a special *defer* action, indicating that prediction should be deferred to a subsequent model.

Definition 1 (Defer tree)**.** *A defer tree T is a binary tree whose leaves are labeled by elements of $\mathcal{Y} \cup \{\text{defer}\}$. For some fallback model B, the prediction of a defer tree is:*

$$\hat{y}_{T,B}(x) = \begin{cases} T(x), & \text{if } T(x) \neq \text{defer}, \\ B(x), & \text{otherwise}. \end{cases}$$

For a fixed fallback model B, we can train a single defer tree T by optimizing (without a certificate of optimality) the objective $\mathcal{L}_{T,B}$ defined by

$$\mathcal{L}_{T,B}(\mathcal{D}^w, \tau, \eta) = \tau\left(|T| - 1\right) + \sum_{i=1}^{N} w_i \left(\mathbb{1}\{\hat{y}_{T,B}(x_i) \neq y_i\} + \eta \mathbb{1}\{T(x_i) = \text{defer}\}\right),$$

where the hyperparameter τ penalizes the number of splits in the tree, η penalizes deferrals to a black box model B, and w denotes the sample weights for dataset $\mathcal{D}$. We optimize this objective using a modified version of LicketySPLIT [5]; details are given in Appendix D.2.

A single defer tree can improve accuracy over a fully interpretable model by deferring difficult samples to a black box. However, those deferred samples still have a problem. On that subset, we are still relying on a black box, and we would like to recover interpretability there as well. It may seem natural to simply grow the tree further on that region, but if the original defer tree is already optimal, this is no longer possible. An optimal defer tree comes with a certificate that rules out further improvement: any attempt to replace a deferral with an interpretable prediction must increase the objective. In particular, matching the black box's accuracy on the deferred subproblems would require a complexity increase whose leaf penalty outweighs the savings from eliminating deferral (formalized in Proposition 2 in the appendix).

To overcome this limitation, we move beyond a single tree and introduce a sequential model that allows further improvement on the deferred region of a single defer tree.

Definition 2 (Multistage defer tree)**.** *A multistage defer tree (MDT) of length K is an ordered sequence*

$$T_{1:k} = (T_1, \ldots, T_K),$$

where each $T_i : \mathcal{X} \to \mathcal{Y} \cup \{\text{defer}\}$ for $i = 1, \ldots, K$ is a defer tree. For some fallback model $B : \mathcal{X} \to \mathcal{Y}$, the prediction of the MDT is

$$\hat{y}_{T_{1:k};B}(x) = \begin{cases} T_1(x), & \text{if } T_1(x) \neq \text{defer}, \\ \hat{y}_{T_{2:k},B}(x), & \text{otherwise}. \end{cases}$$

An MDT routes each input through a sequence of defer trees, where each stage either predicts immediately or defers to the next stage. It also affords interpretability benefits relative to other tree-based ensembles. As discussed in Theorem 1 below, our multistage defer trees can be represented relatively compactly as a rule list. This is significantly smaller than the rule list representations of other ensembles, such as random forests or boosted trees, where even the number of literals in a single antecedent can grow with the sum of depths across all trees (see Proposition 3 in the appendix).

Theorem 1 (Rule-list representation of MDTs)**.** *An MDT with K stages admits a rule-list representation, in which each non-deferral leaf in any stage contributes exactly one antecedent. Consequently, if stage j has n_j non-deferral leaves, the total number of antecedents is $\sum_{j=1}^{K} n_j$.*

Because of Theorem 1, we can define a simple objective for MDTs that corresponds to an upper bound on the rule list deferral objective in Ferry et al. [26] applied to the rule list form of the MDT (we discuss our relation to existing objectives in Appendix E).

For a given MDT $(T_1, \ldots, T_k)$ and fallback model B, let its objective $\mathcal{L}_{T_{1:k},B}$ be:

$$\mathcal{L}_{T_{1:k},B}(\mathcal{D}^w, \tau, \eta) = \tau \sum_{j=1}^{k} \big(|T_j| - 1\big) + \sum_{i=1}^{N} w_i \left(\mathbb{1}\{\hat{y}_{T_{1:k},B}(x_i) \neq y_i\} + \eta \mathbb{1}\{T_{1:k}(x_i) = \text{defer}\}\right),$$

where $T_{1:k}(x)$ denotes the output of the sequence of trees $(T_1, \ldots, T_k)$.

3.1 Training Multistage Defer Trees

When optimizing our MDT objective $\mathcal{L}_{T_{1:k},B}$, we can expand the structure from the top down, training a defer tree at each stage on the remaining deferred region with our single defer tree objective $\mathcal{L}_{T,B}$. When we do so, Proposition 1 shows that we monotonically decrease the objective defined above.

Proposition 1 (Top-down improvement)**.** *Consider an iterative procedure where we train a single defer tree on the samples that have been deferred in all previous stages. Define weights for stage j that zero out all samples that are not deferred by that stage:*

$$w_i^{(j)} := w_i^{(j-1)} \mathbb{1}\{T_{j-1}(x_i) = \text{defer}\}, \quad w_i^{(1)} := w_i.$$

Let Δ_j be how much we improve on the objective of the deferred dataset $\mathcal{D}^{w^{(j)}}$ with defer tree T_j, relative to deferring to a black box B:

$$\Delta_j = \mathcal{L}_{\text{defer},B}(\mathcal{D}^{w^{(j)}}, \tau, \eta) - \mathcal{L}_{T_j,B}(\mathcal{D}^{w^{(j)}}, \tau, \eta).$$

With this iterative procedure, the total improvement of the model relative to the black box is the sum of improvements of each single defer tree relative to deferral.

$$\mathcal{L}_{T_{1:k},B}(\mathcal{D}^w, \tau, \eta) = L_{\text{defer},B}(\mathcal{D}^w, \tau, \eta) - \sum_{j=1}^{k} \Delta_j.$$

While Proposition 1 motivates our approach, there is a risk of a hybrid model deferring to the black box primarily in regions where the black box has overfit. Consider subregions of the space where a black box's performance is especially high due to sample bias. These are also regions where it will be difficult for a simple model to match that performance, so a hybrid model is likely to defer in this case, leading to high test error in deferral cases.

Even for a single defer tree, the generalization of the black-box component can be a substantial issue. Table 2 in the appendix shows that the test error of a black box in the deferred region can be more than an order of magnitude higher than the test error of the simple model in the non-deferred region. To address this, we also need to leverage the rest of the data to improve generalization.

We can continue to refine an MDT without inducing overfitting by reweighting the training distribution: non-deferred points are downweighted based on their distance to the deferred region, rather than assigning them no weight. At a high level, we alternate between fitting a black box and a defer tree and updating the training weights to emphasize points within or near the deferred region.

Let $\mathcal{D}_{k-1} \subseteq \mathcal{X}$ denote the set of deferred points after stage $k-1$. For any sample $x_i \notin \mathcal{D}_{k-1}$, let $\text{defer_dist}_{T_{1:k-1}}(x_i)$ denote the ℓ_1-norm (taken in quantile space, more information in Appendix C) of the minimal change to x_i such that it would be deferred by $T_{1:k-1}$. Using this distance, we define training weights w_i for the next iteration (on the first iteration, all weights are 1).

$$w_i = \begin{cases} 1, & \text{if } x_i \in \mathcal{D}_{k-1} \\ (1-\mu)\,(1 + \text{defer_dist}_{T_{1:k-1}}(x_i))^{-\gamma}, & \text{if } x_i \notin \mathcal{D}_{k-1}. \end{cases}$$

Here, $\mu \in [0, 1]$ controls how much we uniformly scale down the weights of the non-deferred region, while $\gamma \geq 0$ controls how strongly the weights decay with distance from the deferred region. When $\gamma = 0$, this reduces to assigning weight $1 - \mu$ to each non-deferred point. $\mu = 1$ places weight solely on the deferred points.

Algorithm 1 describes our training procedure. At each stage, we fit a fallback model and a defer tree using the current sample weights, initially all 1. If the deferred set decreases in size, we add the new tree to the sequence and update the feasible deferred subregions $\mathcal{S}$ and sample weights w using UPDATESTAGE, described below. Training stops when the deferred set becomes empty or stops shrinking, or when $K_{\max}$ stages have been reached. In the latter case, we retrain the fallback using the final weights. Algorithm 26 in the appendix gives an optional backfitting procedure to

Algorithm 1 Top-down MDT training

Require: Data (X, y), depth d, tree-complexity regularization τ, deferral regularization η, mean reweighting parameter μ, distance downweighting parameter γ, maximum number of stages $K_{\max}$, tree-complexity regularization rescaling flag `rescale_tau`

$\mathcal{T} \leftarrow [\,],\ D \leftarrow \{1, \ldots, N\},\ \mathcal{S} \leftarrow \{\mathbb{R}^p\},\ w \leftarrow (1, 1, \ldots, 1) \in \mathbb{R}^n$ {Each data point starts with a weight of 1; we defer discussion of $\mathcal{S}$ to subsection 3.2}
for $k = 1, \ldots, K_{\max}$ **do**
  $B \leftarrow$ FitFallbackModel(X, y, w) {Train a temporary fallback model; for instance `XGBoost.fit`}
  $T_k \leftarrow$ FitDeferTree$(X, y, B(X), w, \mathcal{S}, \tau, \eta, d)$ {Train the next defer tree pretending it is deferring to B, discussed in subsection 3.3}
  $D' \leftarrow \{i \in D : T_k(x_i) = \text{defer}\}$
  if $D' = D$ **then**
    break
  end if
  $\mathcal{T} \leftarrow \mathcal{T} + [T_k], \quad D \leftarrow D'$
  if $D = \varnothing$ **then**
    break
  end if
  $(\mathcal{S}, w) \leftarrow$ UpdateStage$(X, \mathcal{S}, T_k, D, \mu, \gamma)$ {Get the weights and subregions used in the next stage (discussed in Algorithm 2)}
  if `rescale_tau` **then**
    $\tau \leftarrow \tau \cdot \frac{\sum_i w_i}{N}$ {Rescale regularization for the size of the subproblem}
  end if
  if $k = K_{\max}$ **then**
    $B \leftarrow$ FitFallbackModel(X, y, w) {Train the final fallback model}
  end if
end for
return $\mathcal{T} + [B]$

improve an initial MDT; in practice, we find it is not needed. In Appendix D, we also describe early stopping criteria based on model complexity (e.g., per-stage size, equivalent rule list size, or equivalent single-tree size), along with efficient online updates to enforce these constraints.

We prove additional properties of our method in Appendix A. For instance, we show that the distance-based weights arise as a Gibbs distribution minimizing an entropy-regularized objective.

3.2 Feasible Region Tracking

Let Λ_k^{defer} denote the set of defer leaves of tree k. Each defer leaf $\lambda \in \Lambda_k^{\text{defer}}$ corresponds to an axis-aligned hyperrectangle in feature space, denoted $R(\lambda)$. The total deferral region after stage k can be written as a set of defer subregions in the following way:

$$\mathcal{S}_k := \left(\left\{ \bigcap_{t=1}^{k} R(\lambda_t) \;\middle|\; (\lambda_1, \ldots, \lambda_k) \in \Lambda_1^{\text{defer}} \times \cdots \times \Lambda_k^{\text{defer}} \right\} \right) \setminus \{\emptyset\}.$$

We refer to each element $\bigcap_{t=1}^{k} R(\lambda_t)$ in the set $\mathcal{S}_k$ as a defer subregion. Each such subregion is an axis-aligned hyperrectangle. Many such subregions may be empty (infeasible), e.g., $age \leq 20$ in one stage and $age > 25$ in another; we detect and discard these cases.

Because our iterative procedure refines only the set of subregions that defer, we can update this set iteratively. Algorithm 2 performs this update efficiently and computes the quantities needed for the next stage. We compute the distance from each sample to the deferred region as the minimum distance to any defer subregion. For continuous features, z_j maps each value to quantile space by counting the empirical mass below it plus half the mass at that value. The values L_{sj}, U_{sj} are the boundaries of the defer subregion, snapped inward to the closest observed value, and mapped into quantile space. The axis-aligned and rectangular structure of a defer subregion allows us to efficiently compute the

Algorithm 2 Update stage

Require: $X, \mathcal{S}, T, D, \mu, \gamma$
$\mathcal{S} \leftarrow \{s \cap R(\lambda) \mid s \in \mathcal{S}, \lambda \in \Lambda^{\text{defer}}(T)\} \setminus \{\varnothing\}$
for $i = 1, \ldots, N$ **do**
  $\delta_i \leftarrow \min_{s \in \mathcal{S}} \sum_{j=1}^{p} (\max\{0, L_{sj} - z_j(x_{ij})\} + \max\{0, z_j(x_{ij}) - U_{sj}\})$
  $w_i \leftarrow 1$ if $i \in D$, else $(1-\mu)(1+\delta_i)^{-\gamma}$
end for
return $(\mathcal{S}, w)$

ℓ_1 distance to each subregion by summing, across features, the distance to the nearest value within the subregion's bounds. For features unrestricted by a subregion, we set the quantile-space bounds to $L_{sj} = 0$ and $U_{sj} = 1$, so they contribute zero to the distance. Categorical feature handling is described in Appendix C.

Feasible region tracking can also be used to compress the resulting MDT. In Appendix B, we provide an algorithm to compress the MDT into a single defer tree. We recursively expand each defer leaf by substituting the next-stage tree, while tracking the corresponding subregion of the feature space that we are in. If a split is guaranteed to evaluate the same way throughout this region (i.e., always True or always False), we can replace the node with the corresponding child. We also provide algorithms to further simplify the MDT by eliminating redundant splits; this has the effect of reducing the number

Algorithm 3 DEFERTREE$(Z, y, r, w, h, \tau, \eta)$

Require: Binarized dataset Z, labels y, fallback predictions r, sample weights w, depth h, sparsity penalty τ, deferral penalty η
{Find the best leaf action}
$(T_\ell, C_\ell) \leftarrow$ GREEDY$(Z, y, r, w, 0, \tau, \eta)$
if $h = 0$ **or** $|Z| \leq 1$ **or** $\mathcal{F}(Z) = \varnothing$ **then**
  return (T_ℓ, C_ℓ)
end if
{Evaluate splits using greedy completions}
for $f \in \mathcal{F}(Z)$ **do**
  for $b \in \{0, 1\}$ **do**
    $(Z_b, y_b, r_b, w_b) \leftarrow$ FILTER(Z, y, r, w, f, b)
    $(T^G_{f,b}, C^G_{f,b}) \leftarrow$ GREEDY$(Z_b, y_b, r_b, w_b, h-1, \tau, \eta)$
  end for
  $C^G_f \leftarrow C^G_{f,0} + C^G_{f,1}$
end for
$f^* \leftarrow \arg\min_f C^G_f$
{Recursively fit the selected split}
for $b \in \{0, 1\}$ **do**
  $(Z_b, y_b, r_b, w_b) \leftarrow$ FILTER(Z, y, r, w, f^*, b)
  $(T_b, C_b) \leftarrow$ DEFERTREE$(Z_b, y_b, r_b, w_b, h-1, \tau, \eta)$
end for
{Keep split only if it improves over leaf}
if $C_0 + C_1 < C_\ell$ **then**
  return $(\text{split}(f^*, T_0, T_1), C_0 + C_1)$
end if
return (T_ℓ, C_ℓ)

Algorithm 4 GREEDY$(Z, y, r, w, h, \tau, \eta))$

Require: Binarized features Z, labels y, fallback predictions r, sample weights w, depth h, sparsity penalty τ, deferral penalty η
{Compute the cost of each leaf action}
for each class $c \in \mathcal{Y}$ **do**
  $L_c \leftarrow \tau + \sum_{i=1}^{|Z|} w_i \mathbb{1}\{y_i \neq c\}$
end for
$L_{\text{defer}} \leftarrow \tau + \sum_{i=1}^{|Z|} w_i (\mathbb{1}\{r_i \neq y_i\} + \eta)$
{Find the best leaf action}
$a^* \leftarrow \arg\min_{a \in \mathcal{Y} \cup \{\text{defer}\}} L_a$
$(T_\ell, C_\ell) \leftarrow (\text{leaf}(a^*), L_{a^*})$
if $h = 0$ **or** $|Z| \leq 1$ **or** $\mathcal{F}(Z) = \varnothing$ **then**
  return (T_ℓ, C_ℓ)
end if
{Select the best split according to weighted-entropy}
$f^* \leftarrow$ BESTENTROPYSPLIT(Z, y, w)
{Recursively fit both children}
for $b \in \{0, 1\}$ **do**
  $(Z_b, y_b, r_b, w_b) \leftarrow$ FILTER(Z, y, r, w, f^*, b)
  $(T_b, C_b) \leftarrow$ GREEDY$(Z_b, y_b, r_b, w_b, h-1, \tau, \eta)$
end for
{Keep split only if it improves over leaf}
if $C_0 + C_1 < C_\ell$ **then**
  return $(\text{split}(f^*, T_0, T_1), C_0 + C_1)$
end if
return (T_ℓ, C_ℓ)

of rules or their width in the MDT's rule-list representation. We can apply either of these techniques to each tree in a tree ensemble, using the set of subregions in which that tree may be evaluated.

3.3 Defer Tree Algorithm

Algorithm 3 presents our default procedure for constructing defer trees, which uses the greedy subroutine in Algorithm 4. Both algorithms recursively construct a defer tree on the current subproblem, represented by a binarized dataset Z, labels y, fallback predictions r, and sample weights w, returning a subtree T and its objective value C. Before calling DEFERTREE, FITDEFERTREE can update the binarization at each stage in two ways: by generating new thresholds (e.g., using XGBoost) or by starting with a fixed binarization and removing binary columns that are constant across all feasible defer subregions from a fixed binarization (Appendix B). We compare these approaches against each other and a continuous-feature approach in Appendix G.

Here, $\text{FILTER}(Z, y, r, w, f, b)$ restricts the dataset to samples satisfying $Z_{if} = b$, returning the corresponding subsets of Z, y, r, and w. At each node, we select the best leaf action: predicting a class or deferring to the fallback. We use a per-leaf penalty τ (equivalent in ranking to a per-split penalty). DEFERTREE selects splits using greedy subtree completions before recursively fitting the children, while GREEDY selects splits using weighted label entropy. Both retain a split only if it improves upon the best leaf.

4 Experiments

We compare our MDTs and the defer tree method to state-of-the-art tree ensembles (XGBoost [16], Random Forest [12]), sparse additive tree models (FIGS [72]), and existing hybrid interpretable baselines (HyRS [76, 77], Pre- and Post- CORELS [26]) on a wide range of datasets. We also implemented the hybrid interpretable model from Frost et al. [30], and refer to the method as FLMM. Lastly, we add a logistic regression baseline, which defers when the estimated probability is close to 0.5. Our key questions are as follows. (1) Can MDTs provide accuracy comparable to that of black-box models for a small deferral rate? (2) How do MDT deferral rates and accuracy compare to existing deferral methods? (3) How complex are state-of-the-art MDTs?

To answer these questions, we need to determine the best configurations for hybrid interpretable models subject to a particular constraint on test deferral rate. To do so, we adopt a semi-supervised approach: for each train/test split and maximum deferral rate c, we choose the hyperparameter that maximizes the average validation accuracy, while deferring on at most c proportion of the test set. Note that deferral is independent of labels, so this is fully semi-supervised.

In Appendix G (Tables 3 and 6), we show that the gap between train and test deferral rates is always small, suggesting that a fully supervised selection procedure would yield nearly identical results.

Dataset	Non-Deferral Tree	FIGS	Random Forest	MDT+XGB $\leq$25% (Ours)	XGBoost
Abalone	0.6314 ± 0.0060	$\mathbf{0.6381 \pm 0.0030}$	0.6314 ± 0.0035	0.6333 ± 0.0069	0.6323 ± 0.0070
Adult	0.8598 ± 0.0017	0.8589 ± 0.0021	0.8619 ± 0.0017	0.8667 ± 0.0021	$\mathbf{0.8701 \pm 0.0013}$
Aging	$\mathbf{0.8028 \pm 0.0034}$	0.7944 ± 0.0047	0.8014 ± 0.0036	$\mathbf{0.8028 \pm 0.0034}$	0.8000 ± 0.0028
Bank	0.9051 ± 0.0008	0.9052 ± 0.0017	0.9069 ± 0.0008	0.9087 ± 0.0008	$\mathbf{0.9090 \pm 0.0007}$
Bike	0.9346 ± 0.0021	0.9113 ± 0.0039	0.9426 ± 0.0017	0.9486 ± 0.0024	$\mathbf{0.9500 \pm 0.0023}$
California	0.8893 ± 0.0024	0.8774 ± 0.0015	0.9019 ± 0.0032	0.9099 ± 0.0020	$\mathbf{0.9101 \pm 0.0022}$
Churn	0.9424 ± 0.0036	0.9434 ± 0.0057	0.9506 ± 0.0032	0.9566 ± 0.0028	$\mathbf{0.9574 \pm 0.0028}$
Droid	0.9627 ± 0.0011	0.9601 ± 0.0005	0.9715 ± 0.0006	0.9709 ± 0.0005	$\mathbf{0.9718 \pm 0.0004}$
Heloc	0.7052 ± 0.0107	0.6840 ± 0.0074	0.7080 ± 0.0102	0.7056 ± 0.0107	$\mathbf{0.7116 \pm 0.0108}$
Jasmine	0.8037 ± 0.0077	0.7983 ± 0.0054	$\mathbf{0.8218 \pm 0.0085}$	0.8161 ± 0.0100	0.8214 ± 0.0071
Magic	0.8541 ± 0.0066	0.8539 ± 0.0045	0.8798 ± 0.0059	0.8756 ± 0.0111	0.8849 ± 0.0064
Phishing	0.9528 ± 0.0017	0.9392 ± 0.0019	0.9721 ± 0.0013	0.9728 ± 0.0017	$\mathbf{0.9733 \pm 0.0015}$
Pol	0.9723 ± 0.0012	0.9631 ± 0.0018	0.9823 ± 0.0007	$\mathbf{0.9849 \pm 0.0003}$	0.9845 ± 0.0007
Rl	0.7614 ± 0.0052	0.7163 ± 0.0094	0.8074 ± 0.0058	0.7968 ± 0.0089	$\mathbf{0.8278 \pm 0.0034}$
Shopping	0.9010 ± 0.0025	0.8988 ± 0.0021	0.9046 ± 0.0012	0.9047 ± 0.0016	$\mathbf{0.9057 \pm 0.0015}$
Spambase	0.9254 ± 0.0016	0.9222 ± 0.0034	0.9517 ± 0.0014	0.9500 ± 0.0024	$\mathbf{0.9528 \pm 0.0023}$
Wine	0.8340 ± 0.0015	0.8391 ± 0.0039	0.8848 ± 0.0042	0.8788 ± 0.0052	$\mathbf{0.8864 \pm 0.0022}$

Table 1: Test accuracy (mean $\pm$ standard error across five train/test splits) across representative datasets. Bold indicates the best mean accuracy in each row. Underlining indicates methods whose mean accuracy is within 1.96 standard errors of the best method, corresponding to overlap with the best method's 95% confidence interval.

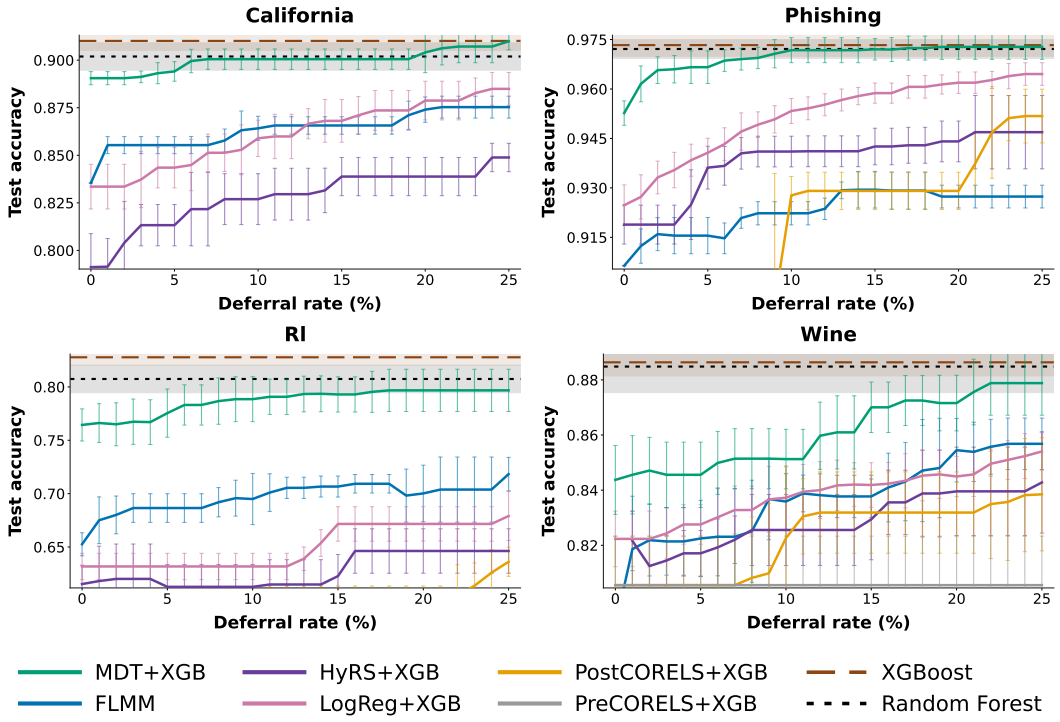


Figure 2: Test accuracy (mean $\pm$ standard deviation) as a function of deferral rate across datasets.

Indeed, we show this is the case in the appendix (Figure 5). Additional information on our experiment setup is in Appendix F; all results are averaged across five train/test splits. The mean and median MDT training times across datasets were 132.1 and 55.4 seconds, respectively (Table 5 in the appendix).

To answer question (1), Table 1 reports test accuracy across datasets. MDT+XGB (with at most 25% deferral) is consistently competitive even with the strongest black box baselines: it is frequently closer to state-of-the-art performance than random forests and almost always within the 95% confidence interval of the best method. Furthermore, it outperforms the sparse ensemble model FIGS on nearly all datasets. MDT also outperforms a single-tree, non-deferral ablation, demonstrating that the multistage training and deferral is core to MDT's success.

To answer question (2), we compare MDTs to other hybrid-interpretable models in Figures 2 and 3. In Figure 2, we compare hybrid-interpretable models on datasets where the accuracy-interpretability tradeoff is high (a black box is $> 2\%$ more accurate than a single tree). On all four datasets, MDT+XGB consistently outperforms all other baselines. On California, MDT+XGB matches the mean accuracy of XGBoost at 25% deferral, a level that no competing method achieves within the same deferral budget. Figure 3 shows the deferral rate required to achieve accuracy within 2.0% of XGBoost across all 25 datasets (the full list of datasets is in Appendix F). At around 40% deferral, our MDT+XGB method can come within this amount of XGBoost for all datasets. No existing method can match this without deferring the majority of the time.

To answer question (3), Figure 4a shows the fraction of datasets that achieve accuracy within 2.0% of XGBoost as a function of the average number of split decisions encountered by a test sample while passing through the interpretable portion of the MDT. This average path-length constraint is enforced analogously to the deferral constraint. At 50% deferral, no dataset requires more than an average of 7.5 split decisions per sample in the interpretable part to reach near-XGBoost performance, and the vast majority reach this threshold with at most an average of 5.5 split decisions per sample. Since MDTs only need to use a black-box component for a small proportion of samples, the black-box representations they learn are naturally more compressed. Figure 4b compares the distribution of the number of splits needed to evaluate samples that are deferred, either by (i) using the final fallback XGBoost model or (ii) using the best XGBoost model for the whole dataset. Both (i) and (ii) are

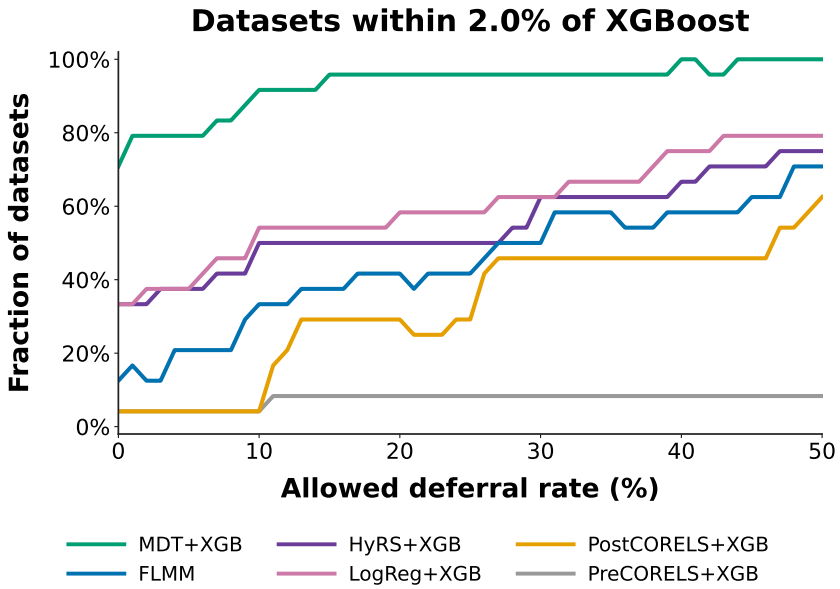
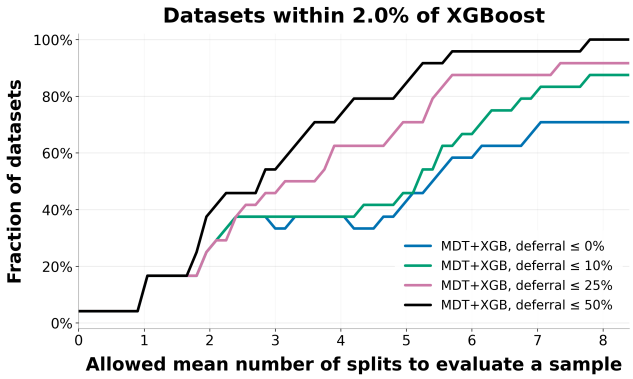


Figure 3: CDF of deferral thresholds required to have accuracy within 2.0% of XGBoost. Each curve shows the fraction of datasets for which a method achieves this accuracy within a given deferral rate.



(a) Fraction of datasets within 2.0% of XGBoost as a function of the average number of split decisions encountered by a test sample in the interpretable portion of MDT+XGB.

Dataset	Splits	Leaves	Δ Acc. 95% CI
Adult	1.021	1.082	$[-0.0048, 0.0012]$
Bike	1.064	1.088	$[-0.0060, 0.0045]$
Compas	1.756	1.545	$[-0.0132, 0.0164]$
Diamonds	1.112	1.458	$[-0.0066, 0.0003]$
Jasmine	1.847	1.361	$[-0.0734, 0.0395]$
Madeline	1.061	11.956	$[-0.0491, 0.0202]$
Phishing	1.808	1.048	$[-0.0029, 0.0024]$
Pol	1.002	1.043	$[-0.0047, 0.0075]$
Rl	1.223	1.327	$[-0.0112, 0.0196]$
Shopping	1.105	1.302	$[-0.0121, 0.0052]$

(b) Mean compression ratios for splits to evaluate a deferred test sample and leaves of the final MDT XGBoost relative to the original XGBoost fallback. A 95% confidence interval on the difference (MDT − XGB) in test deferred region accuracy shows they are indistinguishable. Full table in Table 4 (Appendix).

Figure 4

trained with the same hyperparameter configuration, but approach (i) is trained using a weighting that emphasizes deferred points. The number of splits required to classify a sample is reduced.

5 Conclusion

We introduce Multistage Defer Trees, a model class that adaptively allocates complexity by routing inputs through a sequence of sparse decision trees before deferring to a black-box model. Our training procedure progressively refines the defer region, combining distance-based weighting, which allows the model to borrow strength from nearby points, with compression techniques that track and simplify defer subregions. MDTs demonstrate that accuracy and interpretability need not be traded off globally, substantially improving the accuracy-deferral frontier. Future work could explore using multiple fallback models, possibly from different model classes.

6 Acknowledgments

We acknowledge funding from the National Institutes of Health (NIH) under grant R01-DA054994. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC). Nous remercions le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG) de son soutien. We also thank Zachery Boner for comments on our manuscript, and Varun Babbar for suggestions on earlier directions for the work.

References

[1] In-Vehicle Coupon Recommendation. UCI Machine Learning Repository, 2017. DOI: https://doi.org/10.24432/C5GS4P.

[2] National Poll on Healthy Aging (NPHA). UCI Machine Learning Repository, 2017. DOI: https://doi.org/10.3886/ICPSR37305.v1.

[3] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 3146–3153, 2020.

[4] David Aha. Tic-Tac-Toe Endgame. UCI Machine Learning Repository, 1991. DOI: https://doi.org/10.24432/C5688J.

[5] Varun Babbar, Hayden McTavish, Cynthia Rudin, and Margo Seltzer. Near-optimal decision trees in a SPLIT second. In International Conference on Machine Learning, 2025.

[6] Michelle Bao, Angela Zhou, A. S. Zottola, Brian Brubach, Sarah Desmarais, Seth A. Horowitz, Kristian Lum, and Suresh Venkatasubramanian. It's compaslicated: The messy relationship between rai datasets and algorithmic fairness benchmarks. In Proceedings of the Thirty-Fifth Conference on Neural Information Processing Systems (NeurIPS), 2021. Datasets and Benchmarks Track (Round 1).

[7] Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[8] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. Machine Learning, 106: 1039–1082, 2017.

[9] Guy Blanc, Jane Lange, Chirag Pabbaraju, Colin Sullivan, Li-Yang Tan, and Mo Tiwari. Harnessing the power of choices in decision tree learning. Advances in Neural Information Processing Systems, 36, 2024.

[10] R. Bock. MAGIC Gamma Telescope. UCI Machine Learning Repository, 2004. DOI: https://doi.org/10.24432/C52C8B.

[11] Zachery Boner, Harry Chen, Lesia Semenova, Ronald Parr, and Cynthia Rudin. Using noise to infer aspects of simplicity without learning. In Advances In Neural Information Processing Systems, 2024.

[12] Leo Breiman. Random forests. Machine Learning, 45:5–32, 2001.

[13] Leo Breiman and Nong Shang. Born again trees. University of California, Berkeley, Berkeley, CA, Technical Report, 1(2):4, 1996.

[14] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. Classification and regression trees. wadsworth & brooks, 1984.

[15] Cătălin E Briţa, Jacobus GM van der Linden, and Emir Demirović. Optimal classification trees for continuous feature data using dynamic programming with branch-and-bound. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, pages 11131–11139, 2025.

[16] Tianqi Chen and Carlos Guestrin. Xgboost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 785–794, 2016.

[17] C Chow. On optimum recognition error and reject tradeoff. IEEE Transactions on information theory, 16(1):41–46, 2003.

[18] P. Cortez and A. M. Gonçalves Silva. Using data mining to predict secondary school student performance. 2008. URL https://api.semanticscholar.org/CorpusID:16621299.

[19] Paulo Cortez. Student Performance. UCI Machine Learning Repository, 2008. DOI: https://doi.org/10.24432/C5TG7T.

[20] 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. Journal of Machine Learning Research, 23(26):1–47, 2022.

[21] Emir Demirović, Emmanuel Hebrard, and Louis Jean. Blossom: an anytime algorithm for computing optimal decision trees. In International Conference on Machine Learning, pages 7533–7562. PMLR, 2023.

[22] Ran El-Yaniv et al. On the foundations of noise-free selective classification. Journal of Machine Learning Research, 11(5), 2010.

[23] Nick Erickson, Lennart Purucker, Andrej Tschalzev, David Holzmüller, Prateek Mutalik Desai, David Salinas, and Frank Hutter. Tabarena: A living benchmark for machine learning on tabular data, 2025. URL https://arxiv.org/abs/2506.16791.

[24] Hadi Fanaee-T. Bike Sharing. UCI Machine Learning Repository, 2013. DOI: https://doi.org/10.24432/C5W894.

[25] Hadi Fanaee-T and João Gama. Event labeling combining ensemble detectors and background knowledge. Progress in Artificial Intelligence, 2:113 – 127, 2013. URL https://api.semanticscholar.org/CorpusID:256282956.

[26] Julien Ferry, Gabriel Laberge, and Ulrich Aïvodji. Learning hybrid interpretable models: Theory, taxonomy, and methods. arXiv preprint arXiv:2303.04437, 2023.

[27] FICO. Home equity line of credit (heloc) dataset, 2018. URL https://community.fico.com/s/explainable-machine-learning-challenge. FICO Explainable Machine Learning Challenge.

[28] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1):119–139, 1997. ISSN 0022-0000. doi: https://doi.org/10.1006/jcss.1997.1504. URL https://www.sciencedirect.com/science/article/pii/S002200009791504X.

[29] Yoav Freund, Robert E Schapire, Yoram Singer, and Manfred K Warmuth. Using and combining predictors that specialize. In Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, pages 334–343, 1997.

[30] Nave Frost, Zachary Lipton, Yishay Mansour, and Michal Moshkovitz. Partially interpretable models with guarantees on coverage and accuracy. In International conference on algorithmic learning theory, pages 590–613. PMLR, 2024.

[31] Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michéle Sebag, Alexander Statnikov, WeiWei Tu, and Evelyne Viegas. Analysis of the automl challenge series 2015-2018. In AutoML, Springer series on Challenges in Machine Learning, 2019. URL https://www.automl.org/wp-content/uploads/2018/09/chapter10-challenge.pdf.

[32] Zakk Heile, Hayden McTavish, Varun Babbar, Margo Seltzer, and Cynthia Rudin. From rashomon theory to PRAXIS: Efficient decision tree rashomon sets. In Forty-third International Conference on Machine Learning, 2026. URL https://openreview.net/forum?id=Sgwd0l1u2V.

[33] Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. Nature, 01 2025. doi: 10.1038/s41586-024-08328-6. URL https://www.nature.com/articles/s41586-024-08328-6.

[34] Mark Hopkins, Erik Reeber, George Forman, and Jaap Suermondt. Spambase. UCI Machine Learning Repository, 1999. DOI: https://doi.org/10.24432/C53G6X.

[35] Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. In Advances in Neural Information Processing Systems, volume 32, pages 7265–7273, 2019.

[36] Harold Kiossou, Pierre Schaus, Siegfried Nijssen, and Vinasetan Ratheil Houndji. Time constrained dl8.5 using limited discrepancy search. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 443–459. Springer, 2022.

[37] Harold Kiossou, Pierre Schaus, Siegfried Nijssen, and Gaël Aglin. Efficient lookahead decision trees. In International Symposium on Intelligent Data Analysis, pages 133–144. Springer, 2024.

[38] Harold Silvere Kiossou, Siegfried Nijssen, and Pierre Schaus. A generic complete anytime beam search for optimal decision tree. arXiv preprint arXiv:2508.06064, 2025.

[39] Daniel T Larose and Chantal D Larose. Discovering knowledge in data: an introduction to data mining. John Wiley & Sons, 2015.

[40] Yifan Li, Shuhan Qi, Lei Cui, Chao Xing, Lei Zhang, and Xuan Wang. Interpret when possible: A tree-based hybrid framework for interpretable classification. Big Data Mining and Analytics, 9(1):263–283, 2026. doi: 10.26599/BDMA.2025.9020055. URL https://www.sciopen.com/article/10.26599/BDMA.2025.9020055.

[41] Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. Generalized and scalable optimal sparse decision trees. In International Conference on Machine Learning, pages 6150–6160. PMLR, 2020.

[42] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, 2013. URL https://api.semanticscholar.org/CorpusID:11246170.

[43] David Madras, Toni Pitassi, and Richard Zemel. Predict responsibly: improving fairness and accuracy by learning to defer. Advances in neural information processing systems, 31, 2018.

[44] Preeti N. Malani, Jeffrey Kullgren, and Erica Solway. National poll on healthy aging (npha), united states, april 2017, 2019. URL https://doi.org/10.3886/ICPSR37305.v1. Distributor.

[45] Akshay Mathur. NATICUSdroid (Android Permissions). UCI Machine Learning Repository, 2021. DOI: https://doi.org/10.24432/C5FS64.

[46] Akshay Mathur, Mounika Podila, Keyur Kulkarni, Quamar Niyaz, and Ahmad Y. Javaid. Poster: Naticusdroid: A malware detection framework for android using native and custom. 2021. URL https://api.semanticscholar.org/CorpusID:232063483.

[47] Rahul Mazumder, Xiang Meng, and Haoyue Wang. Quant-BnB: A scalable branch-and-bound method for optimal decision trees with continuous features. In International Conference on Machine Learning, volume 162, pages 15255–15277. PMLR, 17–23 Jul 2022.

[48] Hayden McTavish, Chudi Zhong, Reto Achermann, Ilias Karimalis, Jacques Chen, Cynthia Rudin, and Margo Seltzer. Fast sparse decision tree optimization via reference ensembles. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pages 9604–9613, 2022.

[49] Rami Mohammad and Lee McCluskey. Phishing Websites. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C51W2X.

[50] Rami Mustafa A. Mohammad, Fadi A. Thabtah, and Lee Mccluskey. An assessment of features related to phishing websites using an automated technique. 2012 International Conference for Internet Technology and Secured Transactions, pages 492–497, 2012. URL https://api.semanticscholar.org/CorpusID:5716727.

[51] S. Moro, P. Rita, and P. Cortez. Bank Marketing. UCI Machine Learning Repository, 2014. DOI: https://doi.org/10.24432/C5K306.

[52] Sérgio Moro, P. Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. Decis. Support Syst., 62:22–31, 2014. URL https://api.semanticscholar.org/CorpusID:14181100.

[53] Hussein Mozannar and David Sontag. Consistent estimators for learning to defer to an expert. In International conference on machine learning, pages 7076–7087. PMLR, 2020.

[54] OpenML. Jasmine dataset, 2018. URL https://www.openml.org/d/41143. OpenML Dataset ID 41143; dataset from the ChaLearn Automatic Machine Learning (AutoML) Challenge.

[55] OpenML. Madeline dataset, 2018. URL https://www.openml.org/d/41144. OpenML Dataset ID 41144; dataset from the ChaLearn Automatic Machine Learning (AutoML) Challenge.

[56] OpenML. Diamonds dataset, 2019. URL https://www.openml.org/d/42225. OpenML Dataset ID 42225; dataset containing prices and attributes of nearly 54,000 diamonds.

[57] OpenML. Abalone dataset, 2022. URL https://www.openml.org/d/44956. OpenML Dataset ID 44956; dataset for predicting abalone age from physical measurements.

[58] OpenML. California dataset, 2022. URL https://www.openml.org/d/44090. OpenML Dataset ID 44090; dataset used in the tabular data benchmark and derived from the California Housing dataset.

[59] OpenML. Pol dataset, 2022. URL https://www.openml.org/d/44082. OpenML Dataset ID 44082; dataset used in the tabular data benchmark and derived from a binarized version of the original regression dataset.

[60] OpenML. Rl dataset, 2022. URL https://www.openml.org/d/43949. OpenML Dataset ID 43949; dataset used in the tabular data benchmark and derived from the ChaLearn AutoML Challenge.

[61] OpenML. Wine dataset, 2025. URL https://www.openml.org/d/47041. OpenML Dataset ID 47041.

[62] Danqing Pan, Tong Wang, and Satoshi Hara. Interpretable companions for black-box models. In International conference on artificial intelligence and statistics, pages 2444–2454. PMLR, 2020.

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

[64] J. Ross Quinlan. Induction of decision trees. Machine learning, 1(1):81–106, 1986.

[65] Omer Sagi and Lior Rokach. Approximating xgboost with an interpretable decision tree. Information Sciences, 572:522–542, 2021.

[66] C. Sakar and Yomi Kastro. Online Shoppers Purchasing Intention Dataset. UCI Machine Learning Repository, 2018. DOI: https://doi.org/10.24432/C5F88Q.

[67] Cemal Okan Sakar, Suleyman Olcay Polat, Mete Katircioglu, and Yomi Kastro. Real-time prediction of online shoppers' purchasing intention using multilayer perceptron and lstm recurrent neural networks. Neural Computing and Applications, 31:6893 – 6908, 2018. URL https://api.semanticscholar.org/CorpusID:13682776.

[68] Carla M. Santos-Pereira and Ana M. Pires. On optimal reject rules and roc curves. Pattern Recognition Letters, 26(7):943–952, 2005. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2004.09.042. URL https://www.sciencedirect.com/science/article/pii/S0167865504002892.

[69] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In Proceedings of the Eleventh Annual Conference on Computational Learning Theory, COLT' 98, page 80–91, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 1581130570. doi: 10.1145/279943.279960. URL https://doi.org/10.1145/279943.279960.

[70] Lesia Semenova, Cynthia Rudin, and Ronald Parr. On the existence of simpler machine learning models. In 2022 ACM Conference on Fairness, Accountability, and Transparency, pages 1827–1858, 2022.

[71] Lesia Semenova, Harry Chen, Ronald Parr, and Cynthia Rudin. A path to simpler models starts with noise. Advances in Neural Information Processing Systems, 36, 2023.

[72] Yan Shuo Tan, Chandan Singh, Keyan Nasseri, Abhineet Agarwal, James Duncan, Omer Ronen, Matthew Epland, Aaron Kornblith, and Bin Yu. Fast interpretable greedy-tree sums. Proceedings of the National Academy of Sciences, 122(7):e2310151122, 2025. doi: 10.1073/pnas.2310151122.

[73] Sebastian Thrun. The monk"s problems-a performance comparison of different learning algorithms, cmu-cs-91-197, sch. 1991. URL https://api.semanticscholar.org/CorpusID:59699060.

[74] Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, pages 1625–1632, 2019.

[75] Thibaut Vidal and Maximilian Schiffer. Born-again tree ensembles. In International conference on machine learning, pages 9743–9753. PMLR, 2020.

[76] Tong Wang. Gaining no or low-cost transparency with interpretable partial substitute. In International Conference on Machine Learning, 2019.

[77] Tong Wang and Qihang Lin. Hybrid predictive models: when an interpretable model collaborates with a black-box model. J. Mach. Learn. Res., 22(1), January 2021. ISSN 1532-4435.

[78] Yixiao Wang, Hayden McTavish, Varun Babbar, Margo Seltzer, and Cynthia Rudin. CLARITree: Cholesky and lookahead accelerations for regression with interpretable piecewise linear trees. In Forty-third International Conference on Machine Learning, 2026. URL https://openreview.net/forum?id=JjBozF4i2w.

[79] J. Wnek. MONK's Problems. UCI Machine Learning Repository, 1993. DOI: https://doi.org/10.24432/C5R30R.

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

[81] I-Cheng Yeh and Che hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. Expert Syst. Appl., 36:2473–2480, 2009. URL https://api.semanticscholar.org/CorpusID:15696161.

Appendix Contents

A Proofs . . . 17

Proposition 1: Top-down improvement . . . 17

Proposition 2: Accuracy-deferral trade-off . . . 20

Theorem 1: Rule-list representation of MDTs . . . 21

Corollary 1: MDTs do not have wide rules . . . 22

Proposition 3: Majority-vote ensembles have wide rules . . . 22

Proposition 4: Gibbs form of distance-based weights . . . 22

Proposition 5: Size of the uncompressed single tree representation . . . 24

Proposition 6: Online update for uncompressed subchain size . . . 25

B Compression Details . . . 27

B.1 Subregion Representation . . . 27

B.2 Basic Methods . . . 27

B.3 Compression into a Single Defer Tree . . . 29

B.4 Sequential Compression . . . 30

B.5 Trivial Extensions . . . 33

C Distance to Deferred Region . . . 34

D Full Algorithm . . . 35

D.1 MDT Algorithm . . . 35

D.2 Defer Tree Algorithm . . . 38

D.3 Improving Optimization with Backfitting . . . 42

E Training Objectives . . . 44

F Experiment Setup . . . 46

F.1 Machines . . . 46

F.2 Datasets . . . 46

F.3 Hyperparameter Selection . . . 47

F.4 Cross Validation Procedure . . . 50

G Experimental Results . . . 52

G.1 Ablations . . . 63

H Examples of MDTs . . . 74

Broader Impacts Statement . . . 78

NeurIPS Paper Checklist . . . 79

A Proofs

Proposition 1 (Top-down improvement)**.** *Consider an iterative procedure where we train a single defer tree on the samples that have been deferred in all previous stages. Define weights for stage j that zero out all samples that are not deferred by that stage:*

$$w_i^{(j)} := w_i^{(j-1)} \mathbb{1}\{T_{j-1}(x_i) = \text{defer}\}, \quad w_i^{(1)} := w_i.$$

Let Δ_j be how much we improve on the objective of the deferred dataset $\mathcal{D}^{w^{(j)}}$ with defer tree T_j, relative to deferring to a black box B:

$$\Delta_j = \mathcal{L}_{\text{defer},B}(\mathcal{D}^{w^{(j)}}, \tau, \eta) - \mathcal{L}_{T_j,B}(\mathcal{D}^{w^{(j)}}, \tau, \eta).$$

With this iterative procedure, the total improvement of the model relative to the black box is the sum of improvements of each single defer tree relative to deferral.

$$\mathcal{L}_{T_{1:k},B}(\mathcal{D}^w, \tau, \eta) = L_{\text{defer},B}(\mathcal{D}^w, \tau, \eta) - \sum_{j=1}^{k} \Delta_j.$$

Proof. For each stage j, define the set of samples that reach stage j as

$$D_j := \{i : T_1(x_i) = \cdots = T_{j-1}(x_i) = \text{defer}\},$$

with $D_1 = \{1, \ldots, N\}$.

Making the weight update

$$w_i^{(j)} = w_i \mathbb{1}\{i \in D_j\}.$$

The all-defer baseline has decision tree objective

$$L_{\text{defer},B}(\mathcal{D}^w, \tau, \eta) = \sum_{i=1}^{N} w_i \Big(\mathbb{1}\{B(x_i) \neq y_i\} + \eta \Big),$$

Now consider the contribution of a single stage j on the samples that reach it. On the weighted dataset $\mathcal{D}^{w^{(j)}}$, the single-tree defer objective is

$$L_{T_j,B}(\mathcal{D}^{w^{(j)}}, \tau, \eta) = \tau(|T_j| - 1) + \sum_{i=1}^{N} w_i^{(j)} \Big(\mathbb{1}\{\hat{y}_{T_j,B}(x_i) \neq y_i\} + \eta \mathbb{1}\{T_j(x_i) = \text{defer}\} \Big).$$

The all-defer tree (a single defer leaf) on the same weighted dataset has objective

$$L_{\text{defer},B}(\mathcal{D}^{w^{(j)}}, \tau, \eta) = \sum_{i=1}^{N} w_i^{(j)} \Big(\mathbb{1}\{B(x_i) \neq y_i\} + \eta \Big).$$

Therefore,

$$\Delta_j = L_{\text{defer},B}(\mathcal{D}^{w^{(j)}}, \tau, \eta) - L_{T_j,B}(\mathcal{D}^{w^{(j)}}, \tau, \eta)$$

equals

$$-\tau(|T_j| - 1) + \sum_{i=1}^{N} w_i^{(j)} \Big[\mathbb{1}\{B(x_i) \neq y_i\} + \eta - \mathbb{1}\{\hat{y}_{T_j,B}(x_i) \neq y_i\} - \eta \mathbb{1}\{T_j(x_i) = \text{defer}\} \Big].$$

Now sum the expression for Δ_j over $j = 1, \ldots, k$:

$$\sum_{j=1}^{k} \Delta_j = -\tau \sum_{j=1}^{k} (|T_j| - 1) + \sum_{j=1}^{k} \sum_{i=1}^{N} w_i^{(j)} \Big[\mathbb{1}\{B(x_i) \neq y_i\} + \eta - \mathbb{1}\{\hat{y}_{T_j,B}(x_i) \neq y_i\} - \eta \mathbb{1}\{T_j(x_i) = \text{defer}\} \Big].$$

Using

$$w_i^{(j)} = w_i \mathbb{1}\{T_1(x_i) = \cdots = T_{j-1}(x_i) = \text{defer}\},$$

we can rewrite the double sum sample-by-sample:

$$\sum_{j=1}^{k}\sum_{i=1}^{N} w_i^{(j)}\Big[\mathbb{1}\{B(x_i) \neq y_i\} + \eta - \mathbb{1}\{\hat{y}_{T_j,B}(x_i) \neq y_i\} - \eta\mathbb{1}\{T_j(x_i) = \text{defer}\}\Big]$$

$$= \sum_{i=1}^{N} w_i \sum_{j=1}^{k} \mathbb{1}\{T_1(x_i) = \cdots = T_{j-1}(x_i) = \text{defer}\}\Big[\mathbb{1}\{B(x_i) \neq y_i\} + \eta$$

$$- \mathbb{1}\{\hat{y}_{T_j,B}(x_i) \neq y_i\} - \eta\mathbb{1}\{T_j(x_i) = \text{defer}\}\Big].$$

Fix a sample i. Let r_i be the first stage that predicts a label:

$$r_i := \min\{j : T_j(x_i) \neq \text{defer}\},$$

if such a stage exists. If no such stage exists, set $r_i = k + 1$.

There are 3 cases: (1.1, 1.2, and 2.1).

Case 1.1: $j < r_i \leq k$. Here, sample i reaches stage j but it still defers.

Therefore, $T_j(x_i) = \text{defer}$, so the prediction used in the loss of training the single defer tree is $\hat{y}_{T_j,B}(x_i) = B(x_i)$. Also, $\mathbb{1}\{T_j(x_i) = \text{defer}\} = 1$. So, the bracket becomes

$$\mathbb{1}\{B(x_i) \neq y_i\} + \eta - \mathbb{1}\{B(x_i) \neq y_i\} - \eta = 0.$$

So every stage before the first predicting stage contributes nothing to the improvement in the objective for that sample. Intuitively, before r_i, the model is still doing exactly what the all defer tree does: defers to B and pays the penalty.

Case 1.2: $j = r_i \leq k$. The tree predicts.

We know $T_{r_i}(x_i) \neq \text{defer}$ and $\mathbb{1}\{T_{r_i}(x_i) = \text{defer}\} = 0$. This is the first stage for which we make a prediction for this sample, so $\hat{y}_{T_{r_i},B}(x_i) = \hat{y}_{T_{1:k},B}(x_i)$.

Therefore, the bracket becomes

$$\mathbb{1}\{B(x_i) \neq y_i\} + \eta - \mathbb{1}\{\hat{y}_{T_{1:k},B}(x_i) \neq y_i\}.$$

Since the full MDT does not defer on this sample, we also know $\mathbb{1}\{\hat{T}_{1:k}(x_i) = \text{defer}\} = 0$.

Appending this zero term will become useful later in the proof.

$$\mathbb{1}\{B(x_i) \neq y_i\} + \eta - \Big(\mathbb{1}\{\hat{y}_{T_{1:k},B}(x_i) \neq y_i\} + \eta\mathbb{1}\{\hat{T}_{1:k}(x_i) = \text{defer}\}\Big).$$

Case 2.1: $r_i = k + 1$. Here, the MDT defers to the black box. No stage predicts. The sample is deferred by every stage.

Writing the consequences of this formally:

$$T_1(x_i) = T_2(x_i) = \cdots = T_k(x_i) = \text{defer}.$$

For every $j \leq k$

$$\hat{y}_{T_j,B}(x_i) = B(x_i)$$

and

$$\mathbb{1}\{T_j(x_i) = \text{defer}\} = 1.$$

Therefore, every bracket is

$$\mathbb{1}\{B(x_i) \neq y_i\} + \eta - \mathbb{1}\{B(x_i) \neq y_i\} - \eta = 0.$$

Now, with the three cases solved, we can look back at the double sum.

$$\sum_{j=1}^{k}\sum_{i=1}^{N} w_i^{(j)}\Big[\mathbb{1}\{B(x_i) \neq y_i\} + \eta - \mathbb{1}\{\hat{y}_{T_j,B}(x_i) \neq y_i\} - \eta\mathbb{1}\{T_j(x_i) = \text{defer}\}\Big]$$

$$= \sum_{i=1}^{N} w_i \sum_{j=1}^{k} \mathbb{1}\{T_1(x_i) = \cdots = T_{j-1}(x_i) = \text{defer}\}\Big[\mathbb{1}\{B(x_i) \neq y_i\} + \eta$$

$$- \mathbb{1}\{\hat{y}_{T_j,B}(x_i) \neq y_i\} - \eta\mathbb{1}\{T_j(x_i) = \text{defer}\}\Big].$$

We now claim that S_i is the entire inner sum over stages $j = 1, \ldots, k$ for one sample i:

$$S_i = \mathbb{1}\{B(x_i) \neq y_i\} + \eta - \Big(\mathbb{1}\{\hat{y}_{T_{1:k},B}(x_i) \neq y_i\} + \eta\mathbb{1}\{\hat{T}_{1:k}(x_i) = \text{defer}\}\Big).$$

Recall we showed that at most one j is nonzero for a fixed sample i. If $r_i = k + 1$, every stage defers, so all terms are zero. We appended the zero term in Case 1.2 to give us this expression. The other two cases were evaluated to 0. S_i is precisely the sum of all of them, which simplifies to just Case 1.2.

Continuing, the double sum is

$$= \sum_{i=1}^{N} w_i S_i,$$

$$= \sum_{i=1}^{N} w_i\Big[\mathbb{1}\{B(x_i) \neq y_i\} + \eta - \mathbb{1}\{\hat{y}_{T_{1:k},B}(x_i) \neq y_i\} - \eta\mathbb{1}\{\hat{T}_{1:k}(x_i) = \text{defer}\}\Big].$$

Now plug this back into the sum over improvements,

$$\sum_{j=1}^{k}\Delta_j = -\tau\sum_{j=1}^{k}(|T_j| - 1) + \sum_{i=1}^{N} w_i S_i.$$

$$\sum_{j=1}^{k}\Delta_j = -\tau\sum_{j=1}^{k}(|T_j|-1)+\sum_{i=1}^{N} w_i\Big[\mathbb{1}\{B(x_i) \neq y_i\}+\eta-\mathbb{1}\{\hat{y}_{T_{1:k},B}(x_i) \neq y_i\}-\eta\mathbb{1}\{\hat{T}_{1:k}(x_i) = \text{defer}\}\Big].$$

Rearranging,

$$\sum_{j=1}^{k}\Delta_j = \left[\sum_{i=1}^{N} w_i\big(\mathbb{1}\{B(x_i) \neq y_i\} + \eta\big)\right]$$

$$- \left[\tau\sum_{j=1}^{k}(|T_j| - 1) + \sum_{i=1}^{N} w_i\Big(\mathbb{1}\{\hat{y}_{T_{1:k},B}(x_i) \neq y_i\} + \eta\mathbb{1}\{\hat{T}_{1:k}(x_i) = \text{defer}\}\Big)\right].$$

The second bracket is exactly the MDT objective:

$$\mathcal{L}_{T_{1:k},B}(\mathcal{D}^w, \tau, \eta) = \tau\sum_{j=1}^{k}(|T_j| - 1) + \sum_{i=1}^{N} w_i\Big(\mathbb{1}\{\hat{y}_{T_{1:k},B}(x_i) \neq y_i\} + \eta\mathbb{1}\{\hat{T}_{1:k}(x_i) = \text{defer}\}\Big).$$

Likewise, the first term is the defer leaf objective.

$$L_{\text{defer},B}(\mathcal{D}^w, \tau, \eta) = \sum_i w_i(\mathbb{1}\{B(x_i) \neq y_i\} + \eta).$$

These two facts establish what we wanted to show.

□

Proposition 2 establishes the relationship between sparsity, accuracy, and deferral for sparse decision trees of unbounded depth. Consequently, the same claim holds when the model class is restricted to decision trees of any fixed bounded depth d.

Proposition 2 (Accuracy-deferral trade-off)**.** *Let T^* be an optimal defer tree and let v be a defer leaf. Define $S_v = \{i : x_i \text{ reaches } v\}$.*

For any decision tree U used to replace v, define

$$\Delta(U) = \sum_{i \in S_v} \mathbb{1}\{U(x_i) \neq y_i\} - \sum_{i \in S_v} \mathbb{1}\{B(x_i) \neq y_i\}.$$

Then

$$\Delta(U) \;\geq\; \eta\,|S_v| \;-\; \tau\big(|U| - 1\big).$$

Let $U(x_i)$ In particular, if $U(x_i) = B(x_i)$ for all $i \in S_v$, then

$$|U| \;\geq\; 1 + \frac{\eta\,|S_v|}{\tau}.$$

Proof. We first note that whenever an optimal defer tree places a defer leaf v at a subproblem, then this choice must be at least as good (in objective) as both replacing v with a majority prediction and replacing v with a subtree rooted at that subproblem, regardless of whether the subtree itself contains defer leaves.

The proof is an exchange argument from the optimality of T^*.

The defer-tree objective when referencing a base model with predictions $B(x_i)$ is:

$$L_{\text{defer}}(T) = \tau\big(|T| - 1\big) + \sum_{i=1}^{N} \ell_T(i),$$

where

$$\ell_T(i) = \mathbb{1}\{T(x_i) \in Y,\ T(x_i) \neq y_i\} + \mathbb{1}\{T(x_i) = \text{defer},\ B(x_i) \neq y_i\} + \eta\,\mathbb{1}\{T(x_i) = \text{defer}\}.$$

Take any decision tree U, and form a new defer tree T' by replacing the defer leaf v in T^* with U.

Then, one leaf is removed, and $|U|$ are added.

$$|T'| = |T^*| - 1 + |U|,$$

$$|T'| - |T^*| = |U| - 1.$$

For every $i \notin S_v$, the prediction path is unchanged, so the loss is unchanged.

For every $i \in S_v$, it had previously faced a penalty of $\mathbb{1}\{B(x_i) \neq y_i\} + \eta$, because it was deferred. Now, it faces a cost of $\mathbb{1}\{U(x_i) \neq y_i\}$.

Therefore,

$$L_{\text{defer}}(T') - L_{\text{defer}}(T^*) = \tau\big(|U| - 1\big) + \sum_{i \in S_v} \mathbb{1}\{U(x_i) \neq y_i\} - \sum_{i \in S_v} \big(\mathbb{1}\{B(x_i) \neq y_i\} + \eta\big).$$

Because $L_{\text{defer}}(T') - L_{\text{defer}}(T^*) \geq 0$, we get that

$$\Delta(U) \geq \eta|S_v| - \tau\big(|U| - 1\big).$$

A corollary of this is if $U(x_i) = B(x_i)$, for all $i \in S_v$, then clearly

$$\sum_{i \in S_v} \mathbb{1}\{U(x_i) \neq y_i\} = \sum_{i \in S_v} \mathbb{1}\{B(x_i) \neq y_i\}.$$

Given that $\Delta(U) = 0$, we get $0 \geq \eta|S_v| - \tau\big(|U| - 1\big)$.

Equivalently,

$$|U| \geq 1 + \frac{\eta|S_v|}{\tau}.$$

□

Theorem 1 (Rule-list representation of MDTs)**.** *An MDT with K stages admits a rule-list representation, in which each non-deferral leaf in any stage contributes exactly one antecedent. Consequently, if stage j has n_j non-deferral leaves, the total number of antecedents is $\sum_{j=1}^{K} n_j$.*

Proof. We will use either $|T_j|$ or ℓ_j interchangeably to denote the number of leaves in stage j, and d_j to denote the number of deferral leaves, not to be confused with the depth budget of each tree, which is d.

In stage j, each leaf corresponds to a unique root-to-leaf path, and hence to a conjunction of split conditions. For each non-deferral leaf κ in stage j, let A_κ denote the conjunction of conditions along its path, and let $y_\kappa \in \{0, 1\}$ be its prediction.

We construct a rule list by concatenating K sections, one for each stage. In section j, we include one antecedent (and the corresponding prediction) for each non-deferral leaf κ of stage j:

$$\text{if } A_\kappa \text{ then predict } y_\kappa.$$

Note that A_κ, the conjunction of conditions along a root-to-leaf path in a single-stage tree, contains at most d conditions, since each tree has depth at most d.

Because the leaves of a decision tree partition the input space, and each later stage operates only on the deferred region, the stages cover disjoint parts of the input space. Within a stage, there is no interaction between antecedents, so they may be listed in any order without ambiguity. While the choice of ordering may simplify individual antecedent conditions, it does not change the total number of rules.

We claim this rule list is equivalent to the MDT. Consider any input x. In the MDT, x is first evaluated at stage 1. If it reaches a non-deferral leaf, the MDT outputs the corresponding prediction and halts. In the rule list, exactly the antecedent corresponding to that leaf is satisfied (and the consequent is the label of that leaf), so the first matching rule produces the same output.

If instead x reaches a deferral leaf at stage 1, then no antecedent from section 1 is satisfied, since that section contains only non-deferral leaves. The evaluation proceeds to section 2, mirroring the MDT's transition to stage 2. Repeating this argument inductively, the antecedent that is satisfied corresponds exactly to the non-deferral leaf reached by x in the MDT, and outputs the same prediction.

Thus, the rule list is equivalent to the MDT.

Finally, in stage j, there are $\ell_j - d_j$ non-deferral leaves, each contributing exactly one antecedent. Summing over all stages yields the total number of antecedents:

$$\sum_{j=1}^{K}(\ell_j - d_j).$$

□

Corollary 1 (MDTs do not have wide rules)**.** *Let a rule list be defined over conjunctions of feature queries, where each satisfied rule must immediately output a leaf prediction. Then, for any MDT composed of m defer trees each of depth d, every rule in the equivalent rule-list representation of the interpretable component has at most d literals.*

Proof. This follows direction from the construction discussed in Theorem 1, which uses paths in each tree as rules, each of which must be of depth at most d. □

Proposition 3 (Majority-vote ensembles have wide rules)**.** *Let a rule list be defined over any conjunction of feature queries in a dataset, such that the true branch must be classified immediately with a leaf. Then there exist binary decision trees $T_1, \ldots, T_m$, each of depth d, such that the classifier defined by their majority vote cannot be represented by an equivalent rule list unless some rule queries $\Omega(md)$ literals.*

Proof. Let each tree compute an XOR over d binary features, and suppose that the feature sets used by different trees are disjoint. Assume that all feature value combinations are possible. For an XOR over d features, no subset of features can determine the tree's predictions.

Now consider any rule in an equivalent rule list whose condition is satisfied by at least one input, and that assigns a label. Let us consider using that rule as the first rule. Since the rule must classify the affected points immediately, its conjunction must be sufficient to determine the majority-vote prediction for every completion of the unqueried features.

To force the majority vote, the rule must determine the outputs of at least $\lceil m/2 \rceil$ trees (for the even case, we assume ties can be broken arbitrarily). Because the trees use disjoint feature sets, determining the output of each such XOR tree requires querying all d of its features. Hence, the rule must contain at least

$$d\left(\left\lceil \frac{m}{2} \right\rceil\right) = \Omega(md)$$

literals.

□

As we show in the next theorem, the distance-based weighting rule admits an entropy-regularized variational interpretation. Among all distributions, the normalized weights best balance the expected distance to being deferred and the amount of deviation from uniform (according to γ).

Proposition 4 (Gibbs form of distance-based weights)**.** *For each training data point i, take $c_i := \log(1 + D_i)$ and let u be the uniform distribution over the training data points.*

The following objective (with $\gamma > 0$) has a unique minimizer (over all distributions q on the training data points) given by $q_i \propto (1 + D_i)^{-\gamma}$.

$$\mathcal{J}(q) = \mathbb{E}_q[c_i] + \frac{1}{\gamma}\mathrm{KL}(q\|u)$$

Proof. Plugging in definitions,

$$\mathcal{J}(q) = \sum_{i=1}^{N} q_i c_i + \frac{1}{\gamma}\sum_{i=1}^{N} q_i \log \frac{q_i}{u_i}.$$

Because $u_i = 1/N$,

$$\log \frac{q_i}{u_i} = \log q_i - \log(1/N) = \log q_i + \log N.$$

This yields

$$\mathcal{J}(q) = \sum_{i=1}^{N} q_i c_i + \frac{1}{\gamma} \sum_{i=1}^{N} q_i \log q_i + \frac{1}{\gamma} \sum_{i=1}^{N} q_i \log N = \sum_{i=1}^{N} q_i c_i + \frac{1}{\gamma} \sum_{i=1}^{N} q_i \log q_i + \frac{1}{\gamma} \log N.$$

The third term is constant w.r.t q, so we just need to optimize

$$\sum_{i=1}^{N} q_i c_i + \frac{1}{\gamma} \sum_{i=1}^{N} q_i \log q_i,$$

subject to q being a probability distribution.

We form the Lagrangian:

$$L(q, \lambda) = \sum_{i=1}^{N} q_i c_i + \frac{1}{\gamma} \sum_{i=1}^{N} q_i \log q_i + \lambda \left(\sum_{i=1}^{N} q_i - 1 \right).$$

Because $\frac{d}{dq_i}(q_i \log q_i) = \log q_i + 1$, we get

$$\frac{\partial L}{\partial q_i} = c_i + \frac{1}{\gamma}(\log q_i + 1) + \lambda.$$

For an interior minimizer (assume no $q_i = 0$), this is 0.

$$c_i + \frac{1}{\gamma}(\log q_i + 1) + \lambda = 0.$$

Writing equivalent forms,

$$c_i + \frac{1}{\gamma}(\log q_i + 1) + \lambda = 0.$$

$$\log q_i = -\gamma c_i - 1 - \gamma\lambda.$$

$$q_i = \exp(-1 - \gamma\lambda) \exp(-\gamma c_i).$$

Because $\exp(-1 - \gamma\lambda)$ does not depend on i, we know $q_i \propto e^{-\gamma c_i} = e^{-\gamma c_i} = e^{-\gamma \log(1+D_i)} = (1 + D_i)^{-\gamma}$.

Therefore,

$$q_i \propto (1 + D_i)^{-\gamma}.$$

Normalizing, as we know that q is a probability distribution,

$$q_i = \frac{(1 + D_i)^{-\gamma}}{\sum_{j=1}^{n} (1 + D_j)^{-\gamma}}.$$

We have found the only critical point in the interior ($q_i > 0$). We can show that it is both a minimizer and unique by strict convexity. We have the sum of a linear function (which is convex) and the negative entropy component is strictly convex (with $0 \log 0 := 0$). Since $\gamma > 0$, the sum is strictly convex. This suffices to show that the minimizer is unique and that it is an interior critical point.

□

Proposition 5 (Size of the uncompressed single tree representation)**.** *Let an MDT chain be given by*

$$T_1 \to T_2 \to \cdots \to T_K \to W,$$

where for each stage j, T_j has ℓ_j leaves, of which d_j are deferral leaves, and let W be a fallback model (such as a single-tree representation of a black box) with ℓ_W leaves and no deferral leaves. Then the uncompressed single-tree representation of the chain has

$$\sum_{j=1}^{K} \left(\prod_{i=1}^{j-1} d_i \right) (\ell_j - d_j) + \left(\prod_{i=1}^{K} d_i \right) \ell_W$$

leaves.

Proof. A copy of T_j appears only when the input has been deferred through all earlier stages $T_1, \ldots, T_{j-1}$.

The tree T_1 appears once. Each deferral leaf of T_1 is replaced by a copy of T_2, so there are d_1 copies of T_2. Each deferral leaf of each copy of T_2 is replaced by a copy of T_3, so there are $d_1 d_2$ copies of T_3.

Continuing inductively, the number of copies of T_j in the expanded tree is

$$\prod_{i=1}^{j-1} d_i,$$

with the convention that this product equals 1 when $j = 1$.

Each copy of T_j contributes $\ell_j - d_j$ final (non-deferral) leaves, since its d_j deferral leaves are replaced by the root of the next stage. Therefore, the total number of final leaves contributed by all copies of T_j is

$$\left(\prod_{i=1}^{j-1} d_i \right) (\ell_j - d_j).$$

Summing over all stages $j = 1, \ldots, K$, the total number of final leaves contributed by the MDT stages is

$$\sum_{j=1}^{K} \left(\prod_{i=1}^{j-1} d_i \right) (\ell_j - d_j).$$

If $d_K \neq 0$, the remaining deferral leaves of stage K are replaced by copies of W. The number of copies of T_K is $\prod_{i=1}^{K-1} d_i$, and each such copy has d_K deferral leaves, so the total number of copies of W is

$$\left(\prod_{i=1}^{K-1} d_i \right) d_K = \prod_{i=1}^{K} d_i.$$

Since W has ℓ_W leaves and no deferral leaves, the total number of final leaves contributed by all copies of W is

$$\left(\prod_{i=1}^{K} d_i \right) \ell_W.$$

Adding the disjoint contributions yields the total number of leaves in the uncompressed tree:

$$\sum_{j=1}^{K} \left(\prod_{i=1}^{j-1} d_i \right) (\ell_j - d_j) + \left(\prod_{i=1}^{K} d_i \right) \ell_W.$$

□

To exclude the contribution of W, one may set $d_K = 0$. Alternatively, one may set $\ell_W = 1$; they are equivalent.

Proposition 6 (Online update for uncompressed subchain size)**.** *Let an MDT subchain be given by*

$$T_{K_{\text{lower}}} \to T_{K_{\text{lower}}+1} \to \cdots \to T_{K_{\text{upper}}} \to W,$$

where for each stage j, T_j has ℓ_j leaves, of which d_j are deferral leaves, and let W be a fallback model with ℓ_W leaves and no deferral leaves. Define $p_j := \ell_j - d_j$.

Then the number of leaves in the uncompressed single-tree representation of this subchain is

$$G_W(a,b) = \sum_{j=a}^{b} \left(\prod_{i=a}^{j-1} d_i \right) p_j \; + \; \left(\prod_{i=a}^{b} d_i \right) \ell_W,$$

where $a = K_{\text{lower}}$ and $b = K_{\text{upper}}$.

Moreover, for any $b+1 \le K_{\text{upper}}$ (i.e. whenever stage T_{b+1} exists with total leaf count ℓ_{b+1}, deferral leaf count d_{b+1}, and W' is the new fallback model), this quantity satisfies the following recurrence:

$$\textit{(Top-down)} \quad G_{W'}(a,b+1) = G_W(a,b) + \left(\prod_{i=a}^{b} d_i \right) \left[p_{b+1} + d_{b+1}\ell_{W'} - \ell_W \right],$$

where W' denotes the retrained fallback model after extending the chain.

Proof. The expression for $G_W(a,b)$ follows from the same counting argument as for the full chain. A copy of T_j appears only when the input has been deferred through all earlier stages $T_a, \ldots, T_{j-1}$, yielding $\prod_{i=a}^{j-1} d_i$ copies. Each copy contributes $p_j = \ell_j - d_j$ final leaves, since deferral leaves are replaced by the next stage. The final deferrals from stage b produce $\prod_{i=a}^{b} d_i$ copies of W, each contributing ℓ_W leaves.

For the recurrence, extending the chain by appending T_{b+1} and retraining the fallback model to W' yields

$$G_{W'}(a,b+1) = \sum_{j=a}^{b+1} \left(\prod_{i=a}^{j-1} d_i \right) p_j + \left(\prod_{i=a}^{b+1} d_i \right) \ell_{W'}.$$

Separating the $j = b+1$ term,

$$= \sum_{j=a}^{b} \left(\prod_{i=a}^{j-1} d_i \right) p_j + \left(\prod_{i=a}^{b} d_i \right) p_{b+1} + \left(\prod_{i=a}^{b+1} d_i \right) \ell_{W'}.$$

Using the definition of $G_W(a,b)$,

$$G_W(a,b) = \sum_{j=a}^{b} \left(\prod_{i=a}^{j-1} d_i \right) p_j + \left(\prod_{i=a}^{b} d_i \right) \ell_W,$$

we subtract and add appropriately to obtain

$$G_{W'}(a,b+1) = G_W(a,b) - \left(\prod_{i=a}^{b} d_i \right) \ell_W + \left(\prod_{i=a}^{b} d_i \right) p_{b+1} + \left(\prod_{i=a}^{b+1} d_i \right) \ell_{W'}.$$

Factoring $\prod_{i=a}^{b} d_i$ and using

$$\prod_{i=a}^{b+1} d_i = \left(\prod_{i=a}^{b} d_i \right) d_{b+1},$$

gives

$$G_{W'}(a,b+1) = G_W(a,b) + \left(\prod_{i=a}^{b} d_i \right) \left[p_{b+1} + d_{b+1}\ell_{W'} - \ell_W \right].$$

This establishes the desired recurrence. □

In our algorithm, taking $\ell_W = \ell_{W'} = 1$ (counting the leaves of deferring to the black box, but not the black box complexity) is helpful in online updates for early stopping.

At each stage, we update this quantity using the top-down recurrence, treating each current deferral path as contributing one leaf. This yields an incremental update of the form

$$G \leftarrow G + \left(\prod_{i=a}^{b} d_i \right) (\ell_{b+1} - 1),$$

which can be computed efficiently using running quantities that are already maintained for other early-stopping criteria based on the sequential model complexity. If this estimate exceeds a specified leaf budget, we terminate training early.

B Compression Details

B.1 Subregion Representation

A subregion s is represented by two kinds of constraints. For each numerical or ordinal feature f, the subregion may store an interval

$$x_f \in (\ell_f, u_f],$$

where absent bounds are interpreted as $(-\infty, \infty]$. For each one-hot categorical group g (of size m_g), the subregion stores a vector

$$a_g \in \{-1, 0, 1\}^{m_g},$$

where $a_{gh} = 1$ means category h is forced to be active, $a_{gh} = -1$ means category h is ruled out, and $a_{gh} = 0$ means its status is still unknown. We use these constraints to determine whether a split is already implied by the current subregion, and to refine the subregion when a new branch literal is added. For instance, we know that no category can be ruled out, and that there cannot be more than 1 active category.

B.2 Basic Methods

ADDLITERAL. The ADDLITERAL routine refines a subregion s by incorporating a new branching constraint $\ell = (f, \nu, b)$. This is done in two steps. For continuous or ordinal variables, TIGHTENNUMERICALBOUNDS updates the interval for feature f by intersecting it with the constraint induced by the branch. For categorical variables, DEDUCECATEGORICALIMPLICATIONS updates any categorical group constraints implied by this literal. Together, these operations produce a new subregion s' that represents the intersection of the original region with the branch condition.

Algorithm 5 ADDLITERAL(s, ℓ)

Require: Subregion s, literal $\ell = (f, \nu, b)$, where $b = \texttt{true}$ denotes $f \leq \nu$
Ensure: Refined subregion s'
$s' \leftarrow$ TIGHTENNUMERICALBOUNDS(s, ℓ)
$s' \leftarrow$ DEDUCECATEGORICALIMPLICATIONS(s', ℓ)
return s'

TIGHTENNUMERICALBOUNDS. This routine updates the interval constraint for a numerical (or ordinal) feature f. If the literal corresponds to taking the True branch ($f \leq \nu$), the upper bound is tightened to $\min\{u_f, \nu\}$. If the literal corresponds to the False branch ($f > \nu$), the lower bound is tightened to $\max\{\ell_f, \nu\}$.

Algorithm 6 TIGHTENNUMERICALBOUNDS$(s, (f, \nu, b))$

Require: Subregion s, literal (f, ν, b)
Ensure: Subregion with updated numerical interval for f
$s' \leftarrow$ COPY(s)
$(\ell_f, u_f] \leftarrow s'.\text{bounds}(f)$, using $(-\infty, \infty]$ if absent
if $b = \texttt{true}$ **then**
  $u_f \leftarrow \min\{u_f, \nu\}$
else
  $\ell_f \leftarrow \max\{\ell_f, \nu\}$
end if
$s'.\text{bounds}(f) \leftarrow (\ell_f, u_f]$
return s'

DEDUCECATEGORICALIMPLICATIONS. This routine handles the case where features are part of a one-hot categorical group. For binary decision trees, split thresholds of the form $f \leq \nu$ force the feature to be 0, while $f > \nu$ forces it to be 1. We encode this by setting the corresponding entry in the group vector to -1 (inactive) or 1 (active), respectively.

Multiple categories may be inactive (ruled out, set to -1), but at most one category can be active (set to 1). If any entry is set to 1, all other entries in the group must be set to -1. Conversely, if all but one entry are -1, the remaining entry must be 1, since exactly one category must be active.

Algorithm 7 DEDUCECATEGORICALIMPLICATIONS$(s, (f, \nu, b))$

Require: Subregion s, literal (f, ν, b)
Ensure: Subregion with updated categorical constraints
$s' \leftarrow \text{COPY}(s)$
if f is not a one-hot categorical feature **then**
  return s'
end if
if ν is not a binary split threshold **then**
  return s'
end if
Let g be the categorical group containing feature f
Let h be the position of feature f within group g
Initialize $a_g \in \{-1, 0, 1\}^{m_g}$ to all zeros if absent
if $b =$ `true` **then**
  $a_{gh} \leftarrow -1$ $\{f \leq \nu$ means this one-hot feature is inactive, consider $\nu = \frac{1}{2}\}$
else
  $a_{gh} \leftarrow 1$ $\{f > \nu$ means this one-hot feature is active, consider $\nu = \frac{1}{2}\}$
end if
if there exists h such that $a_{gh} = 1$ **then**
  Set $a_{gr} \leftarrow -1$ for all $r \neq h$
else if exactly one entry h has $a_{gh} = 0$ **then**
  $a_{gh} \leftarrow 1$
end if
return s'

IMPLIESTRUE. The routine IMPLIESTRUE determines whether a split condition $f \leq \nu$ is guaranteed to hold for all possible points in the subregion s. For numerical features, this occurs when the upper bound of the interval satisfies $u_f \leq \nu$, meaning every feasible point lies on the True side. For categorical features, $f \leq \nu$ corresponds to the feature being 0. Thus, if the categorical status is -1 (meaning the feature is forced to be 0), the condition is always true. Otherwise, it is either always forced to be 1, or we do not know, so the outcome is not implied, and the routine returns false.

Algorithm 8 IMPLIESTRUE(s, f, ν)

Require: Subregion s, split $f \leq \nu$
Ensure: Whether any hypothetical point in s satisfies $f \leq \nu$
$(\ell_f, u_f] \leftarrow s.\text{bounds}(f)$, using $(-\infty, \infty]$ if absent
if $u_f \leq \nu$ **then**
  return `true`
end if
if f is a one-hot categorical feature and ν is a binary split threshold **then**
  Let a_{gh} be the status of feature f in its categorical group
  if $a_{gh} = -1$ **then**
    return `true`
  end if
end if
return `false`

IMPLIESFALSE. The routine IMPLIESFALSE determines whether a split condition $f \leq \nu$ is guaranteed to fail for all possible points in the subregion s. For numerical features, this occurs when the lower bound satisfies $\ell_f \geq \nu$, meaning all feasible points lie strictly on the False side. For categorical features, $f \leq \nu$ is always false precisely when the feature equals 1.

Algorithm 9 IMPLIESFALSE(s, f, ν)

Require: Subregion s, split $f \leq \nu$
Ensure: Whether any hypothetical point in s satisfies $f > \nu$
$(\ell_f, u_f] \leftarrow s.\text{bounds}(f)$, using $(-\infty, \infty]$ if absent
if $\ell_f \geq \nu$ **then**
  return `true`
end if
if f is a one-hot categorical feature and ν is a binary split threshold **then**
  Let a_{gh} be the status of feature f in its categorical group
  if $a_{gh} = 1$ **then**
    return `true`
  end if
end if
return `false`

B.3 Compression into a Single Defer Tree

A compressed single-tree representation can substantially reduce root-to-leaf path lengths by specializing later-stage trees to the particular subregion in which they are evaluated. For example, a later-stage split such as `age` ≤ 23 may help distinguish inputs in subregions with $18 <$ `age` ≤ 25 and `age` > 25, but is redundant for any path already restricted to `age` ≤ 18; when unrolling the MDT into a single tree, such splits can be pruned along the branches where they have no effect. On a high level, we will do this by putting the tree of stage $i + 1$ in the defer leaves of stage i, simplifying it, and repeating once we reach the new defer leaves.

Algorithm 10 constructs a compressed single-tree representation of the full MDT by recursively expanding the staged decision process into one tree.

Algorithm 10 BUILDCOMPRESSEDSINGLETREE$(v, k, s, R_1, \ldots, R_K)$

Require: Node v, stage index k, subregion s, stage roots $R_1, \ldots, R_K$
Ensure: Compressed subtree rooted at v under subregion s
if v is a leaf **then**
  if $v.action \neq$ `defer` **then**
    return leaf with action $v.action$
  end if
  $k' \leftarrow k + 1$
  if $k' \leq K$ **then**
    return BUILDCOMPRESSEDSINGLETREE$(R_{k'}, k', s, R_1, \ldots, R_K)$
  end if
  return leaf with action `black_box`
end if
$f \leftarrow v.feature$
$\nu \leftarrow v.threshold$
if IMPLIESTRUE(s, f, ν) **then**
  return BUILDCOMPRESSEDSINGLETREE$(v.left, k, s, R_1, \ldots, R_K)$
end if
if IMPLIESFALSE(s, f, ν) **then**
  return BUILDCOMPRESSEDSINGLETREE$(v.right, k, s, R_1, \ldots, R_K)$
end if
$s_L \leftarrow$ ADDLITERAL$(s, (f, \nu,$ `true`$))$
$s_R \leftarrow$ ADDLITERAL$(s, (f, \nu,$ `false`$))$
$u_L \leftarrow$ BUILDCOMPRESSEDSINGLETREE$(v.left, k, s_L, R_1, \ldots, R_K)$
$u_R \leftarrow$ BUILDCOMPRESSEDSINGLETREE$(v.right, k, s_R, R_1, \ldots, R_K)$
return internal node with split (f, ν), left child u_L, right child u_R

The procedure takes as input a node v from stage k, along with a subregion s describing the set of inputs that may reach this node. The subregion s encodes all constraints accumulated along the path taken so far, potentially across multiple stages.

If v is a leaf, there are two cases. If the leaf corresponds to a terminal prediction (i.e., not `defer`), then the prediction is returned directly. If the leaf corresponds to `defer`, then we transition to the next stage: we recurse on the root of stage $k + 1$ with the same subregion s. If no further stages remain, we return a `black_box` leaf, representing the final fallback model.

If v is an internal node with split (f, ν), we again leverage the subregion s to simplify the structure. If s provably implies that all possible inputs would satisfy the split (i.e., always take the True branch), then we recurse only on the left child. Similarly, if s implies the False branch, we recurse only on the right child. In either case, the split can be removed because it does not affect any possible point in the subregion.

Otherwise, the split is ambiguous under s, and both branches must be explored. We construct refined subregions s_L and s_R by adding the corresponding literal to s (whether we are supposing we take the True or False branch), thereby capturing the additional constraint imposed by each branch. The algorithm then recursively builds the left and right subtrees under these refined subregions and returns a new internal node with these children.

B.4 Sequential Compression

In this section, we show how to (possibly) further compress a sequential representation of the MDT. Because the points outside the defer region are downweighted, it is somewhat unlikely that there exist splits in the MDT that provably do not separate any possible pair of points in the defer region; however, it is still possible. We note that compression of the sequential representation is exactly the compression done to reduce the number of literals in the rules of the rule list representation.

Algorithm 11 simplifies a tree relative to a set of possible incoming subregions $\mathcal{S}$ (i.e., multiple defer leaves may lead to it, not just one). Intuitively, compared to constructing a single tree from the MDT, we must evaluate each split with respect to all subregions simultaneously, rather than a single input region.

Algorithm 11 SIMPLIFYSEQUENTIAL$(v, \mathcal{S})$

Require: Node v, incoming set of subregions $\mathcal{S}$
Ensure: Compressed subtree rooted at v
1: **if** v is a leaf **then**
2: **return** leaf with action $v.action$
3: **end if**
4: $f \leftarrow v.feature$
5: $\tau \leftarrow v.threshold$
6: $all_true \leftarrow$ `true`
7: $all_false \leftarrow$ `true`
8: **for all** $s \in \mathcal{S}$ **do**
9: **if** not IMPLIESTRUE(s, f, τ) **then**
10: $all_true \leftarrow$ `false`
11: **end if**
12: **if** not IMPLIESFALSE(s, f, τ) **then**
13: $all_false \leftarrow$ `false`
14: **end if**
15: **end for**
16: **if** all_true **then**
17: **return** SIMPLIFYSEQUENTIAL$(v.left, \mathcal{S})$
18: **end if**
19: **if** all_false **then**
20: **return** SIMPLIFYSEQUENTIAL$(v.right, \mathcal{S})$
21: **end if**
22: $\mathcal{S}_L \leftarrow [\,]$
23: $\mathcal{S}_R \leftarrow [\,]$
24: **for all** $s \in \mathcal{S}$ **do**
25: **if** not IMPLIESFALSE(s, f, τ) **then**
26: $s' \leftarrow$ TIGHTENNUMERICALBOUNDS$(s, (f, \tau,$ `true`$))$
27: $s' \leftarrow$ DEDUCECATEGORICALIMPLICATIONS$(s', (f, \tau,$ `true`$))$
28: append s' to $\mathcal{S}_L$
29: **end if**
30: **if** not IMPLIESTRUE(s, f, τ) **then**
31: $s' \leftarrow$ TIGHTENNUMERICALBOUNDS$(s, (f, \tau,$ `false`$))$
32: $s' \leftarrow$ DEDUCECATEGORICALIMPLICATIONS$(s', (f, \tau,$ `false`$))$
33: append s' to $\mathcal{S}_R$
34: **end if**
35: **end for**
36: $u_L \leftarrow$ SIMPLIFYSEQUENTIAL$(v.left, \mathcal{S}_L)$
37: $u_R \leftarrow$ SIMPLIFYSEQUENTIAL$(v.right, \mathcal{S}_R)$
38: **return** internal node with split (f, τ), left child u_L, right child u_R

The procedure is recursive and begins at the root node. For each subregion $s \in \mathcal{S}$, we determine whether the split always routes every possible input consistent with s (not just training samples, but any hypothetical point that satisfies the subregion constraints) to the True branch, the False branch, or whether the outcome is ambiguous (lines 8–15). If and only if every subregion is guaranteed to take the True branch, then ALL_TRUE is set to True. Similarly, if and only if every subregion is guaranteed to take the False branch, then ALL_FALSE is set to True. In either of these cases, the split can be pruned, and we recurse on the corresponding child. Otherwise, the split cannot be removed, and we must recurse on both branches.

In lines 24–34, we construct two (potentially overlapping) collections of subregions before recursing:

$$\mathcal{S}_L = \left\{ s^{\leq} \;:\; s \in \mathcal{S},\ s \text{ does not provably imply } f > t,\ s^{\leq} \text{ is } s \text{ refined with } f \leq t \right\},$$

$$\mathcal{S}_R = \left\{ s^{>} \;:\; s \in \mathcal{S},\ s \text{ does not provably imply } f \leq t,\ s^{>} \text{ is } s \text{ refined with } f > t \right\}.$$

Each collection is formed by selecting subregions that are compatible with the corresponding branch and refining them with the constraint induced by the split. Subregions that already contradict a branch are excluded from that side.

These two collections are not disjoint. Any subregion s that does not imply either outcome will appear in both sets (after being refined by the respective branch on which we recurse).

Algorithm 11 simplifies a tree as much as we provably can in this traversal when given a set of subregions that the tree will be used on. However, we still need to shrink this set of subregions before simplifying the next tree in the MDT.

Algorithm 12 Build compressed stage trees

Require: Stage roots $R_1, \ldots, R_K$
Ensure: Compressed stage roots $\widetilde{R}_1, \ldots, \widetilde{R}_K$
1: $\mathcal{S} \leftarrow [\,]$
2: $\widetilde{\mathcal{R}} \leftarrow [\,]$
3: **for** $k = 1, \ldots, K$ **do**
4: $\quad \widetilde{R}_k \leftarrow \textsc{SimplifySequential}(R_k, \mathcal{S})$
5: $\quad$ append $\widetilde{R}_k$ to $\widetilde{\mathcal{R}}$
6: $\quad \mathcal{P} \leftarrow \textsc{CollectPathsToAction}(\widetilde{R}_k, \texttt{defer})$
7: $\quad \mathcal{S} \leftarrow \textsc{ExtendSubregionsWithFeasiblePaths}(\mathcal{S}, \mathcal{P})$
8: **end for**
9: **return** $\widetilde{\mathcal{R}}$

Algorithm 12 performs this update iteratively across stages. At stage k, after simplifying the tree R_k into $\widetilde{R}_k$, we identify all root-to-leaf paths that result in a `defer` decision. Each such path $p \in \mathcal{P}$ corresponds to a conjunction of literals describing a region of the input space that will be deferred by the current stage. To construct the subregions for the next stage, we combine the incoming subregions $\mathcal{S}$ with these defer paths. Concretely, Algorithm 13 forms a Cartesian product between $\mathcal{S}$ and $\mathcal{P}$: for each subregion $s \in \mathcal{S}$ and each defer path $p \in \mathcal{P}$, we attempt to extend s by sequentially incorporating the literals along p. This process can be viewed as intersecting the region represented by s with the region defined by the path p. By adding one literal at a time, we allow for early termination. A contradiction occurs if a literal violates the existing numerical bounds of the subregion or is incompatible with previously deduced categorical constraints. If any such violation occurs, the extended subregion is discarded. Otherwise, we tighten the numerical bounds and propagate any implied categorical constraints, producing a refined subregion c.

The resulting set $\mathcal{S}_{next}$ therefore consists precisely of those subregions that are simultaneously: (i) reachable under the previous stages (captured by $\mathcal{S}$), and (ii) routed to deferral by the current stage (captured by $\mathcal{P}$).

Algorithm 13 EXTENDSUBREGIONSWITHFEASIBLEPATHS$(\mathcal{S}, \mathcal{P})$

Require: Incoming subregions $\mathcal{S}$, defer paths $\mathcal{P}$
Ensure: Next-stage incoming subregions
$\mathcal{S}_{next} \leftarrow [\,]$
for all $s \in \mathcal{S}$ **do**
  for all $p \in \mathcal{P}$ **do**
    $c \leftarrow s$
    $feasible \leftarrow$ `true`
    for all literal $\ell \in p$ **do**
      if VIOLATESNUMERICBOUNDS(c, ℓ) **then**
        $feasible \leftarrow$ `false`
        break
      end if
      if VIOLATESCATEGORICALCONSTRAINTS(c, ℓ) **then**
        $feasible \leftarrow$ `false`
        break
      end if
      $c \leftarrow$ TIGHTENNUMERICALBOUNDS(c, ℓ)
      $c \leftarrow$ DEDUCECATEGORICALIMPLICATIONS(c, ℓ)
    end for
    if $feasible$ **then**
      append c to $\mathcal{S}_{next}$
    end if
  end for
end for
return $\mathcal{S}_{next}$

B.5 Trivial Extensions

After recursively simplifying the left and right subtrees of a split, we may encounter a situation where both children reduce to identical leaf nodes with the same action. In this case, the split no longer affects the prediction for any input and can be safely removed. We refer to such splits as trivial extensions and perform this pruning as a post-order traversal at the end of any compression scheme.

Algorithm 14 PRUNETRIVIALEXTENSIONS(v)

Require: Node v
Ensure: Tree rooted at v with trivial extensions removed
if v is a leaf **then**
  return leaf with action $v.action$
end if
$u_L \leftarrow$ PRUNETRIVIALEXTENSIONS$(v.left)$
$u_R \leftarrow$ PRUNETRIVIALEXTENSIONS$(v.right)$
if u_L and u_R are leaves and $u_L.action = u_R.action$ **then**
  return leaf with action $u_L.action$
end if
return internal node with split $(v.feature, v.threshold)$, left child u_L, right child u_R

C Distance to Deferred Region

Each feasible subregion $s \in \mathcal{S}_{k-1}$ stores the constraints induced by the current deferred region. Let $\mathcal{J}_{\text{num}}(s)$ denote the set of continuous (or ordinal) features that have been constrained in subregion s, and let $\mathcal{J}_{\text{cat}}(s)$ denote the set of categorical feature groups that have been constrained. For each $j \in \mathcal{J}_{\text{num}}(s)$, let $(\ell_{sj}, u_{sj}]$ denote the interval induced by subregion s. For each $j \in \mathcal{J}_{\text{cat}}(s)$, let A_{sj} denote the set of categories consistent (allowed by) with subregion s. Additional details on handling categorical variables are provided in Appendix B. At a high level, for a one-hot encoded categorical group, a subregion may rule out a category via a False split, leave it unconstrained, or identify it as active either through a True split or by eliminating all other categories. The categories which are consistent with the subregion are exactly the categories that are not ruled out by the subregion.

We first map each continuous or ordinal feature into its empirical quantile coordinate so that distances across coordinates are on a common scale. Since features may be discrete, we use the mid-rank empirical CDF, assigning each value v the quantile

$$z_j(v) = \frac{\#\{X_j < v\} + \frac{1}{2}\#\{X_j = v\}}{N}.$$

Because interval endpoints may not coincide with observed feature values (e.g., midpoints), we evaluate endpoint quantiles by snapping to the nearest realized value that remains within the interval. Specifically, define

$$\underline{z}_j(t) = \inf\{z_j(v) : v > t,\ v \in \text{supp}(X_j)\}, \qquad \overline{z}_j(t) = \sup\{z_j(v) : v \le t,\ v \in \text{supp}(X_j)\}.$$

Then, for each training point x_i that is not deferred, define its distance to the current deferred region by

$$\text{defer_dist}_{T_{\text{l:k-1}}}(x_i) = \min_{s \in \mathcal{S}_{k-1}} \left(\sum_{j \in \mathcal{J}_{\text{num}}(s)} d_j^{(s)}(x_i) + \sum_{j \in \mathcal{J}_{\text{cat}}(s)} d_j^{(s)}(x_i) \right).$$

For continuous (or ordinal) features $j \in \mathcal{J}_{\text{num}}(s)$, we measure distance in quantile space as

$$d_j^{(s)}(x_i) = \max\{\underline{z}_j(\ell_{sj}) - z_j(x_{ij}), 0\} + \max\{z_j(x_{ij}) - \overline{z}_j(u_{sj}), 0\}.$$

For categorical features $j \in \mathcal{J}_{\text{cat}}(s)$, let c_{ij} denote the category of point x_i on feature j. We define

$$d_j^{(s)}(x_i) = \begin{cases} 0, & c_{ij} \in A_{sj}, \\ \frac{1}{2}, & c_{ij} \notin A_{sj}. \end{cases}$$

This choice is consistent with treating a binary feature as continuous under the mid-rank empirical CDF, where a mismatch incurs a quantile distance of $\frac{1}{2}$; we extend this to general categorical features.

Unconstrained features (i.e., $j \notin \mathcal{J}_{\text{num}}(s) \cup \mathcal{J}_{\text{cat}}(s)$) contribute zero distance as no value for them will change what leaf you are in.

Using these distances, we convert each $\text{defer_dist}_{T_{\text{l:k-1}}}(x_i)$ into a similarity score

$$(1 + \text{defer_dist}_{T_{\text{l:k-1}}}(x_i))^{-\gamma}.$$

Let $\mathcal{D}_{k-1} \subseteq \mathcal{X}$ denote the set of deferred points after stage $k-1$. We define training weights w_i for the next iteration (on the first iteration, all weights are 1).

$$w_i \ = \ \begin{cases} 1, & \text{if } x_i \in \mathcal{D}_{k-1} \\ (1-\mu)\,(1 + \text{defer_dist}_{T_{\text{l:k-1}}}(x_i))^{-\gamma}, & \text{if } x_i \notin \mathcal{D}_{k-1} \end{cases}$$

When $\gamma = 0$, this reduces to assigning a uniform weight of $1 - \mu$ to each non-deferred point and a weight of 1 to each deferred point.

D Full Algorithm

D.1 MDT Algorithm

Algorithm 15 presents the full version of our algorithm. There are two additions relative to the algorithm presented in the main paper. The first is that we update the set of subregions $\mathcal{S}$ (lines 4, 21, 22, 28) and pass it into our FILTERSPLITS and GETWEIGHTS so it does not have to be recomputed from scratch each iteration. The second is that we additionally support early termination of training if the model complexity becomes too large. This is measured in 3 ways: (i) a stage of the tree has too many leaves, (ii) the sum of leaves across stages becomes too large, or (iii) the number of leaves in the naïve representation of an MDT as a single tree becomes too large. (iii) takes advantage of a recurrence derived in Proposition 6. Given that we have already computed that tree, we choose to add it to MDT anyway, stopping after that.

Algorithm 15 Top-down MDT training with feasible-region tracking

Require: Data $(X, X_{\text{binarized}}, y)$ with $X \in \mathbb{R}^{N\times p}$ and $X_{\text{binarized}} \in \{0,1\}^{N\times q}$, depth d, τ, η, $\mu \geq 0$, boolean flag rescale_tau, maximum number of stages $K_{\max}$, $\gamma \geq 0$, optional budgets $L_{\text{stage}}, L_{\text{total}}, L_{\text{expanded}}$
Ensure: MDT stages and fallback model
1: $B \leftarrow$ FITFALLBACKMODEL$(X, y, \mathbf{1})$
2: $\mathcal{T}_{\text{interp}} \leftarrow [\,]$
3: $D \leftarrow \{1, \ldots, N\}$
4: $\mathcal{S} \leftarrow \{s_\emptyset\}$ {Initial unconstrained subregion}
5: $\Pi \leftarrow 1$ {Running product of deferral leaves}
6: $G \leftarrow 1$ {Expanded-leaf count with fallback counted as one leaf}
7: **for** $k = 1, \ldots, K_{\max}$ **do**
8: **if** $k = 1$ **then**
9: $T_k \leftarrow$ FITDEFERTREE$(X_{\text{binarized}}, y, B(X), \mathbf{1}, \tau, \eta, d)$
10: **else**
11: $X_{\text{filtered}} \leftarrow$ FILTERSPLITS$(X_{\text{binarized}}, \mathcal{S})$
12: $w \leftarrow$ GETWEIGHTS$(X, D, \mathcal{S}, \gamma, \mu)$
13: $B \leftarrow$ FITFALLBACKMODEL(X, y, w)
14: $\tau_k \leftarrow \tau \cdot \frac{\sum_{i=1}^{N} w_i}{N}$ **if** rescale_tau **else** τ
15: $T_k \leftarrow$ FITDEFERTREE$(X_{\text{filtered}}, y, B(X), w, \tau_k, \eta, d)$
16: **end if**
17: $\mathcal{T}_{\text{interp}} \leftarrow \mathcal{T}_{\text{interp}} + [T_k]$
18: $\ell_k \leftarrow$ NUMLEAVES(T_k)
19: $d_k \leftarrow$ NUMDEFERLEAVES(T_k)
20: $D_{\text{new}} \leftarrow \{i \in D : T_k(x_i) = \texttt{defer}\}$
21: $\mathcal{P} \leftarrow$ COLLECTPATHSTOACTION$(T_k, \texttt{defer})$
22: $\mathcal{S}_{\text{new}} \leftarrow$ EXTENDSUBREGIONSWITHFEASIBLEPATHS$(\mathcal{S}, \mathcal{P})$
23: $(\texttt{should_stop}, G, \Pi) \leftarrow$ SHOULDSTOPTRAINING$(D, D_{\text{new}}, \ell_k, d_k, \mathcal{T}_{\text{interp}}, G, \Pi, L_{\text{stage}}, L_{\text{total}}, L_{\text{expanded}})$
24: **if** should_stop **then**
25: **break**
26: **end if**
27: $D \leftarrow D_{\text{new}}$
28: $\mathcal{S} \leftarrow \mathcal{S}_{\text{new}}$
29: **end for**
30: **if** $D \neq \varnothing$ **then**
31: $w \leftarrow$ GETWEIGHTS$(X, D, \mathcal{S}, \gamma, \mu)$
32: $B \leftarrow$ FITFALLBACKMODEL(X, y, w)
33: **end if**
34: **return** $\mathcal{T}_{\text{interp}} + [B]$

Algorithm 16 includes the aforementioned stopping criteria. It terminates training if the deferred set becomes empty, fails to shrink, or if any complexity budget is exceeded. The expanded tree

size is tracked online using a recurrence, allowing early stopping without constructing a single tree representation.

Algorithm 16 SHOULDSTOPTRAINING$(D, D_{\text{new}}, \ell_k, d_k, \mathcal{T}_{\text{interp}}, G, \Pi, L_{\text{stage}}, L_{\text{total}}, L_{\text{expanded}})$

Require: Previous deferred set D, new deferred set D_{new}, current tree leaf count ℓ_k, current defer-leaf count d_k, current stage list $\mathcal{T}_{\text{interp}}$, running expanded count G, running defer-product Π, optional budgets $L_{\text{stage}}, L_{\text{total}}, L_{\text{expanded}}$
Ensure: Boolean `should_stop`, updated G, updated Π
if $D_{\text{new}} = \varnothing$ **then**
  return (**true**, G, Π) {No points remain deferred}
end if
if $D_{\text{new}} = D$ **then**
  return (**true**, G, Π) {Deferred set did not shrink}
end if
if L_{expanded} is specified **then**
  $G \leftarrow G + \Pi(\ell_k - 1)$
  $\Pi \leftarrow \Pi d_k$
end if
if L_{stage} is specified and $\ell_k > L_{\text{stage}}$ **then**
  return (**true**, G, Π) {Current stage exceeds per-tree budget}
end if
if L_{total} is specified and $\sum_{T \in \mathcal{T}_{\text{interp}}} |T| > L_{\text{total}}$ **then**
  return (**true**, G, Π) {Sequential model exceeds total leaf budget}
end if
if L_{expanded} is specified and $G > L_{\text{expanded}}$ **then**
  return (**true**, G, Π) {Expanded single-tree representation exceeds budget}
end if
return (**false**, G, Π)

Algorithm 17 removes binary features that do not distinguish between feasible deferred subregions. A feature is discarded if it is provably always true or always false across all subregions (note that we can't discard a feature if it is provably true on half of the subregions and provably false on the others). This has the added bonus of reducing the search space, which is beneficial for near-optimal decision tree algorithms. We implement this by storing, for each column, a string representation of the split (e.g., `age<=25`), which we parse to recover the feature and threshold. The column itself in the binarized dataset includes which samples satisfy the split.

Algorithm 17 FILTERSPLITS($X_{\text{binarized}}, \mathcal{S}$)

Require: Binarized feature matrix $X_{\text{binarized}}$, feasible deferred subregions $\mathcal{S}$
Ensure: Filtered binarized feature matrix

$I \leftarrow [\,]$
for $j = 1, \ldots, q$ **do**
  Get the split $(f \leq \nu)$ from the metadata associated with column j
  `all_true` $\leftarrow$ **true**
  `all_false` $\leftarrow$ **true**
  for all $s \in \mathcal{S}$ **do**
    if not IMPLIESTRUE(s, f, ν) **then**
      `all_true` $\leftarrow$ **false**
    end if
    if not IMPLIESFALSE(s, f, ν) **then**
      `all_false` $\leftarrow$ **false**
    end if
  end for
  if not `all_true` and not `all_false` **then**
    append j to I
  end if
end for
return $X_{\text{binarized}}[:, I]$

Algorithm 18 assigns weights based on distance to the deferred region. Deferred points receive full weight, while non-deferred points are down-weighted according to their proximity to the nearest feasible subregion. This emphasizes points near the deferral boundary during training.

Algorithm 18 GETWEIGHTS$(X, D, \mathcal{S}, \gamma, \mu)$

Require: Data X, current deferred index set D, feasible deferred subregions $\mathcal{S}$, decay parameter γ, non-deferred downweighting parameter μ
Ensure: Weight vector $w \in \mathbb{R}^N_{\geq 0}$

for $i = 1, \ldots, N$ **do**
  if $i \in D$ **then**
    $w_i \leftarrow 1$
  else
    $\Delta_i \leftarrow \min_{s \in \mathcal{S}}$ DISTANCETOSUBREGION(x_i, s)
    $w_i \leftarrow (1 - \mu)(1 + \Delta_i)^{-\gamma}$
  end if
end for
return w

Algorithm 19 computes the distance from a point to a subregion by summing per-feature deviations. Numerical features are measured in quantile space, while categorical features incur a fixed penalty of $\frac{1}{2}$ if the subregion does not allow the category that the point is in (i.e, the path to the subregion took a split of false on that category). Unconstrained features contribute zero.

Algorithm 19 DISTANCETOSUBREGION(x_i, s)

Require: Point x_i, feasible subregion s
Ensure: Distance from x_i to s
$\Delta \leftarrow 0$
for all numerical or ordinal features $j \in \mathcal{J}_{\text{num}}(s)$ **do**
  Let $(\ell_{sj}, u_{sj}]$ be the interval for feature j in s
  $\Delta \leftarrow \Delta + \max\{\underline{z}_j(\ell_{sj}) - z_j(x_{ij}), 0\} + \max\{z_j(x_{ij}) - \overline{z}_j(u_{sj}), 0\}$
end for
for all categorical groups $j \in \mathcal{J}_{\text{cat}}(s)$ **do**
  Let A_{sj} be the set of categories allowed by s
  Let c_{ij} be the category of x_i in group j
  if $c_{ij} \notin A_{sj}$ **then**
    $\Delta \leftarrow \Delta + \frac{1}{2}$
  end if
end for
return Δ

D.2 Defer Tree Algorithm

In Algorithm 20 construct each defer tree using a recursive, top-down procedure that jointly optimizes over prediction and deferral decisions. This algorithm uses the same heuristics to select splits as are used in the LicketySPLIT algorithm of [5]. Our implementation extends the efficient C++ codebase of Heile et al. [32], which implements a modified version of LicketySPLIT. Their algorithm considers an unweighted dataset and builds standard decision trees (without deferral). Our changes lie in supporting weights and deferral, thus optimizing a different objective. We solve the optimization problem using a per-leaf penalty, which is equivalent to optimizing a per-split penalty, since the number of leaves in a decision tree is always exactly one more than the number of splits, differing only by an additive constant τ.

In our code, we also modify two other algorithms so that they can construct defer trees: the greedy decision tree algorithm implemented in scikit-learn [63], which operates directly on continuous features, and a modified version of CLARITree [78].

Algorithm 20, at any node, considers three possible leaf actions: predicting class 0, predicting class 1, or deferring to the fallback model. Each action is assigned an objective value that combines a fixed leaf penalty τ with a weighted misclassification cost.

At a high level, we will recursively consider every possible split. We query a greedy method to build a subtree on the left and right child of every split, and choose the split that minimizes our objective under greedy completions. After choosing this split, we recurse on the left and right children of this split, and repeat.

Algorithm 20 FITDEFERTREE$(X_{\text{binarized}}, y, r, w, \tau, \eta, d)$

Require: Binarized data $X_{\text{binarized}}$, labels y, fallback predictions r, weights w, leaf penalty τ, defer penalty η, depth budget d
Ensure: A defer tree T
$A \leftarrow \{1, \ldots, N\}$
$(T, C) \leftarrow$ BUILDDEFERTREE(A, d)
return T

The greedy subroutine (Algorithm 22) follows a very similar structure: it considers all splits and chooses the one that minimizes the weighted label entropy of the resulting partition (we minimize this quantity in Algorithm 23).

Algorithm 23 selects splits by minimizing the weighted label entropy of the resulting partition. This focuses on separating the labels rather than directly optimizing the defer-aware objective. As this is the standard heuristic in greedy decision tree algorithms, we adopt it here.

Algorithm 21 BUILDDEFERTREE(A, d)

Require: Index set A, remaining depth d
Ensure: A pair (T, C), where T is a defer subtree on A and C is its objective value
$W_A \leftarrow \sum_{i \in A} w_i$
$C_0 \leftarrow \tau + \sum_{i \in A} w_i \mathbf{1}\{y_i = 1\}$
$C_1 \leftarrow \tau + \sum_{i \in A} w_i \mathbf{1}\{y_i = 0\}$
$C_{\text{defer}} \leftarrow \tau + \sum_{i \in A} w_i \mathbf{1}\{r_i \neq y_i\} + \eta W_A$
$(C_{\text{leaf}}, a_{\text{leaf}}) \leftarrow \min\{(C_0, 0), (C_1, 1), (C_{\text{defer}}, \texttt{defer})\}$
if $d = 0$ or $|A| \leq 1$ **then**
  return (leaf with action $a_{\text{leaf}}, C_{\text{leaf}}$)
end if
$f^\star \leftarrow \perp$
$C^\star \leftarrow \infty$
for $f = 1, \ldots, q$ **do**
  $A_L \leftarrow \{i \in A : X_{\text{binarized},if} = 1\}$
  $A_R \leftarrow \{i \in A : X_{\text{binarized},if} = 0\}$
  if $A_L = \varnothing$ or $A_R = \varnothing$ **then**
    continue
  end if
  $(\widetilde{T}_L, \widetilde{C}_L) \leftarrow$ GREEDY$(A_L, d-1)$
  $(\widetilde{T}_R, \widetilde{C}_R) \leftarrow$ GREEDY$(A_R, d-1)$
  $C_f \leftarrow \widetilde{C}_L + \widetilde{C}_R$
  if $C_f < C^\star$ **then**
    $C^\star \leftarrow C_f$
    $f^\star \leftarrow f$
  end if
end for
if $f^\star = \perp$ **then**
  return (leaf with action $a_{\text{leaf}}, C_{\text{leaf}}$)
end if
$A_L^\star \leftarrow \{i \in A : X_{\text{binarized},if^\star} = 1\}$
$A_R^\star \leftarrow \{i \in A : X_{\text{binarized},if^\star} = 0\}$
$(T_L, C_L) \leftarrow$ BUILDDEFERTREE$(A_L^\star, d-1)$
$(T_R, C_R) \leftarrow$ BUILDDEFERTREE$(A_R^\star, d-1)$
$C_{\text{split}} \leftarrow C_L + C_R$
if $C_{\text{leaf}} \leq C_{\text{split}}$ **then**
  return (leaf with action $a_{\text{leaf}}, C_{\text{leaf}}$)
end if
Get the non-binarized feature f_{orig} and threshold ν (in $\leq \nu$) from the metadata of column $f^\star$
$T \leftarrow$ internal node with $T.\texttt{feature} = f_{\text{orig}}$, $T.\texttt{threshold} = \nu$, left child T_L, and right child T_R
return (T, C_{split})

Algorithm 22 GREEDY(A, d)

Require: Index set A, remaining depth d
Ensure: A pair (T, C), where T is a greedy defer subtree on A and C is its objective value

$W_A \leftarrow \sum_{i \in A} w_i$
$C_0 \leftarrow \tau + \sum_{i \in A} w_i \mathbf{1}\{y_i = 1\}$
$C_1 \leftarrow \tau + \sum_{i \in A} w_i \mathbf{1}\{y_i = 0\}$
$C_{\text{defer}} \leftarrow \tau + \sum_{i \in A} w_i \mathbf{1}\{r_i \neq y_i\} + \eta W_A$
$(C_{\text{leaf}}, a_{\text{leaf}}) \leftarrow \min\{(C_0, 0), (C_1, 1), (C_{\text{defer}}, \texttt{defer})\}$
if $d = 0$ or $|A| \leq 1$ **then**
  return (leaf with action $a_{\text{leaf}}, C_{\text{leaf}}$)
end if
$f^\star \leftarrow$ BESTENTROPYSPLIT(A)
if $f^\star = \perp$ **then**
  return (leaf with action $a_{\text{leaf}}, C_{\text{leaf}}$)
end if
$A_L^\star \leftarrow \{i \in A : X_{\text{binarized},if^\star} = 1\}$
$A_R^\star \leftarrow \{i \in A : X_{\text{binarized},if^\star} = 0\}$
if $A_L^\star = \varnothing$ or $A_R^\star = \varnothing$ **then**
  return (leaf with action $a_{\text{leaf}}, C_{\text{leaf}}$)
end if
$(T_L, C_L) \leftarrow$ GREEDY$(A_L^\star, d-1)$
$(T_R, C_R) \leftarrow$ GREEDY$(A_R^\star, d-1)$
$C_{\text{split}} \leftarrow C_L + C_R$
if $C_{\text{leaf}} \leq C_{\text{split}}$ **then**
  return (leaf with action $a_{\text{leaf}}, C_{\text{leaf}}$)
end if
$T \leftarrow$ internal node splitting on $X_{\text{binarized},f^\star} = 1$, with left child T_L and right child T_R
return (T, C_{split})

Algorithm 23 BESTENTROPYSPLIT(A)

Require: Index set A
Ensure: Feature index $f^\star$ minimizing weighted entropy (or $\perp$ if none valid)

$f^\star \leftarrow \perp$
$H^\star \leftarrow \infty$
for $f = 1, \ldots, q$ **do**
  $A_L \leftarrow \{i \in A : X_{\text{binarized},if} = 1\}$
  $A_R \leftarrow \{i \in A : X_{\text{binarized},if} = 0\}$
  if $A_L = \varnothing$ or $A_R = \varnothing$ **then**
    continue
  end if
  $W_L \leftarrow \sum_{i \in A_L} w_i$
  $W_R \leftarrow \sum_{i \in A_R} w_i$
  $W \leftarrow W_L + W_R$
  if $W = 0$ **then**
    continue
  end if
  $p_L \leftarrow \begin{cases} \dfrac{\sum_{i \in A_L} w_i y_i}{W_L}, & W_L > 0 \\ 0, & \text{otherwise} \end{cases}$
  $p_R \leftarrow \begin{cases} \dfrac{\sum_{i \in A_R} w_i y_i}{W_R}, & W_R > 0 \\ 0, & \text{otherwise} \end{cases}$
  $H_L \leftarrow \begin{cases} -p_L \log p_L - (1 - p_L) \log(1 - p_L), & p_L \in (0, 1) \\ 0, & \text{otherwise} \end{cases}$
  $H_R \leftarrow \begin{cases} -p_R \log p_R - (1 - p_R) \log(1 - p_R), & p_R \in (0, 1) \\ 0, & \text{otherwise} \end{cases}$
  $H_f \leftarrow \dfrac{W_L}{W} H_L + \dfrac{W_R}{W} H_R$
  if $H_f < H^\star$ **then**
    $H^\star \leftarrow H_f$
    $f^\star \leftarrow f$
  end if
end for
return $f^\star$

D.3 Improving Optimization with Backfitting

The MDT training procedure constructs the model as a sequence of defer trees that are trained with respect to a temporary black box fallback model (Algorithms 1 and 15). At the first stage, we train T_1 to either predict or defer to an initial fallback model B_1. After this stage identifies a remaining deferred region, we reweight the data toward that region, train a new fallback B_2, and then train T_2 to either predict or defer to B_2. Continuing this process, we train T_i to defer to B_i, but in the final model, T_i does not necessarily defer to B_i. For instance, if there are 4 trees in the MDT, T_1 defers to another MDT, not a black box.

Therefore, once full MDT has been constructed, it may be beneficial to retrain stage j to defer to the entire suffix $T_{j+1:K}; B$. This motivates the following procedure: choose a component of the MDT, hold the rest of the model fixed, and retrain that component against the suffix that follows it. As we are optimizing using near-optimal algorithms, it will also be helpful to have an acceptance step to accept the new defer tree only if it is better in the objective that the algorithm tried to optimize.

The backward pass (Algorithm 24) starts from the final fallback model and moves upstream through the MDT, retraining each stage conditioned on the rest of it. We then visit stages in reverse order (optionally starting with the fallback model) doing exactly that. In contrast, the forward pass (Algorithm 25) does exactly the same thing but it starts from the first stage. We put these together in an alternating approach (Algorithm 26). It combines these two sweeps in the following update order: $B,\ T_K, T_{K-1}, \ldots, T_1,\ T_2, T_3, \ldots, T_K, \ldots$. The results of using this approach are shown in Figure 14. In short, backfitting gives clear improvements on some datasets, such as Tictactoe, where revisiting stages may help capture higher-order interactions, but it degrades performance on several others.

Algorithm 24 BACKWARDPASS: Backfitting MDT stages in reverse order

Require: Data $(X, X_{\text{binarized}}, y)$, MDT stages $\mathcal{T}_{\text{interp}} = [T_1, \ldots, T_K]$, fallback model B, depth d, τ, η, $\mu \geq 0$, $\gamma \geq 0$, boolean flag rescale_tau, endpoints j_{start} and j_{end}
Ensure: Updated MDT stages and fallback model

for $j = j_{\text{start}}, j_{\text{start}} - 1, \ldots, j_{\text{end}}$ **do**
  $D_{j-1} \leftarrow \{i : T_1(x_i) = \cdots = T_{j-1}(x_i) = \texttt{defer}\}$
  $\mathcal{S}_{j-1} \leftarrow$ GETDEFERREDSUBREGIONS$(T_{1:j-1})$
  $w \leftarrow$ GETWEIGHTS$(X, D_{j-1}, \mathcal{S}_{j-1}, \gamma, \mu, T_{1:j-1})$
  $\widehat{y}^{\text{suffix}} \leftarrow \widehat{y}_{T_{j+1:K}, B}(X)$
  $\tau_j \leftarrow \tau \cdot \frac{\sum_{i=1}^{N} w_i}{N}$ **if** rescale_tau **else** τ
  $X_{\text{filtered}} \leftarrow$ FILTERSPLITS$(X_{\text{binarized}}, \mathcal{S}_{j-1})$
  $\widetilde{T}_j \leftarrow$ FITDEFERTREE$(X_{\text{filtered}}, y, \widehat{y}^{\text{suffix}}, w, \tau_j, \eta, d)$
  $L_{\text{old}} \leftarrow L_{T_j, T_{j+1:K}; B}(D^w, \tau_j, \eta)$
  $L_{\text{new}} \leftarrow L_{\widetilde{T}_j, T_{j+1:K}; B}(D^w, \tau_j, \eta)$
  if $L_{\text{new}} \leq L_{\text{old}}$ **then**
    $T_j \leftarrow \widetilde{T}_j$
  end if
end for
return $[T_1, \ldots, T_K] + [B]$

Algorithm 25 FORWARDPASS: Backfitting MDT stages in forward order

Require: Data $(X, X_{\text{binarized}}, y)$, MDT stages $\mathcal{T}_{\text{interp}} = [T_1, \ldots, T_K]$, fallback model B, depth d, τ, η, $\mu \geq 0$, $\gamma \geq 0$, boolean flag rescale_tau, endpoints j_{start} and j_{end}
Ensure: Updated MDT stages and fallback model

for $j = j_{\text{start}}, j_{\text{start}} + 1, \ldots, j_{\text{end}}$ **do**
  $D_{j-1} \leftarrow \{i : T_1(x_i) = \cdots = T_{j-1}(x_i) = \texttt{defer}\}$
  $\mathcal{S}_{j-1} \leftarrow$ GETDEFERREDSUBREGIONS$(T_{1:j-1})$
  $w \leftarrow$ GETWEIGHTS$(X, D_{j-1}, \mathcal{S}_{j-1}, \gamma, \mu, T_{1:j-1})$
  $\widehat{y}^{\text{suffix}} \leftarrow \widehat{y}_{T_{j+1:K}, B}(X)$
  $\tau_j \leftarrow \tau \cdot \frac{\sum_{i=1}^{N} w_i}{N}$ **if** rescale_tau **else** τ
  $X_{\text{filtered}} \leftarrow$ FILTERSPLITS$(X_{\text{binarized}}, \mathcal{S}_{j-1})$
  $\widetilde{T}_j \leftarrow$ FITDEFERTREE$(X_{\text{filtered}}, y, \widehat{y}^{\text{suffix}}, w, \tau_j, \eta, d)$
  $L_{\text{old}} \leftarrow L_{T_j, T_{j+1:K}; B}(D^w, \tau_j, \eta)$
  $L_{\text{new}} \leftarrow L_{\widetilde{T}_j, T_{j+1:K}; B}(D^w, \tau_j, \eta)$
  if $L_{\text{new}} \leq L_{\text{old}}$ **then**
    $T_j \leftarrow \widetilde{T}_j$
  end if
end for
return $[T_1, \ldots, T_K] + [B]$

Algorithm 26 Backfitting for MDTs

Require: Data $(X, X_{\text{binarized}}, y)$, MDT stages $\mathcal{T}_{\text{interp}} = [T_1, \ldots, T_K]$, fallback model B, number of sweeps M, depth d, τ, η, $\mu \geq 0$, $\gamma \geq 0$, boolean flag rescale_tau
Ensure: Updated MDT stages and fallback model

for $m = 1, \ldots, M$ **do**
  $D_K \leftarrow \{i : T_1(x_i) = \cdots = T_K(x_i) = \texttt{defer}\}$
  $\mathcal{S}_K \leftarrow$ GETDEFERREDSUBREGIONS$(T_{1:K})$
  $w \leftarrow$ GETWEIGHTS$(X, D_K, \mathcal{S}_K, \gamma, \mu, T_{1:K})$
  $B \leftarrow$ FITFALLBACKMODEL(X, y, w)
  $\mathcal{T}_{\text{interp}} + [B] \leftarrow$ BACKWARDPASS$(X, X_{\text{binarized}}, y, \mathcal{T}_{\text{interp}}, B, d, \tau, \eta, \mu, \gamma, \text{rescale_tau}, K, 1)$
  $\mathcal{T}_{\text{interp}} + [B] \leftarrow$ FORWARDPASS$(X, X_{\text{binarized}}, y, \mathcal{T}_{\text{interp}}, B, d, \tau, \eta, \mu, \gamma, \text{rescale_tau}, 2, K)$
end for
return $\mathcal{T}_{\text{interp}} + [B]$

E Training Objectives

Learning a sparse decision tree is commonly formulated as minimizing a regularized empirical risk of the form

$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{1}\{T(x_i)\neq y_i\}+\lambda|T|,$$

where $|T|$ is the number of leaves in the tree [5, 41, 48]. For binary trees, the number of internal split nodes is always $|T|-1$. Therefore, minimizing a penalty on leaves is equivalent, up to an additive constant λ, to minimizing a penalty on split nodes:

$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{1}\{T(x_i)\neq y_i\}+\lambda|T|=\frac{1}{N}\sum_{i=1}^{N}\mathbb{1}\{T(x_i)\neq y_i\}+\lambda(|T|-1)+\lambda.$$

Since the two objectives differ only by a constant term, they have the same set of minimizers. We choose to work with the number of splits for a reason that will become clear in the MDT objective.

Hybrid interpretable rule-list models instead optimize an objective that balances the error of the full hybrid system, the complexity of the interpretable rule list, and the amount of deferral to a black-box model. In the notation of Ferry et al. [26], for a rule list r and black box B, this objective has the form

$$\frac{\widehat{L}_{\mathcal{D}}(B,r)}{N}+\lambda|r|+\beta\frac{|\mathcal{D}\setminus\mathcal{D}_r|}{N},$$

where $\widehat{L}_{\mathcal{D}}(B,r)$ is the empirical error of the full hybrid model, $|r|$ is the number of rules, and $\mathcal{D}_r$ is the subset of samples captured by the rule list. Equivalently, $|\mathcal{D}\setminus\mathcal{D}_r|/N$ is the empirical deferral rate.

Our single defer tree objective combines these two perspectives. It is a sparse decision tree objective combined with a soft penalty for deferral:

$$\mathcal{L}_{T,B}(\mathcal{D}^w,\tau,\eta)=\tau(|T|-1)+\sum_{i=1}^{N}w_i\left(\mathbb{1}\{\hat{y}_{T,B}(x_i)\neq y_i\}+\eta\mathbb{1}\{T(x_i)=\text{defer}\}\right).$$

When all weights are $w_i=1$ and $\eta=0$, we recover the same loss (except we do not normalize by the dataset size N) as [5, 41, 48], with $\tau=\lambda N$.

The MDT objective is then the natural extension of the single defer-tree objective: it penalizes the predictions and deferral rate after routing through all stages of the MDT.

$$\mathcal{L}_{T_{1:k},B}(\mathcal{D}^w,\tau,\eta)=\tau\sum_{j=1}^{k}(|T_j|-1)+\sum_{i=1}^{N}w_i\left(\mathbb{1}\{\hat{y}_{T_{1:k},B}(x_i)\neq y_i\}+\eta\mathbb{1}\{T_{1:k}(x_i)=\text{defer}\}\right).$$

When $k=1$, this reduces exactly to the single defer-tree objective. For MDTs, it is more natural to regularize the number of splits rather than leaves because if there was a stage with a single defer leaf, regularizing by the number of leaves would increase the MDT objective, though nothing was changed about the predictions or the amount that the MDT deferred.

Finally, Theorem 1 shows that any MDT has an associated rule-list representation with

$$|r_{\text{MDT}}|=\sum_{j=1}^{k}(|T_j|-d_j),$$

where $|T_j|$ is the number of leaves of stage j and d_j is the number of deferral leaves in that stage. Assume the MDT has a non-zero deferral rate to the fallback model: this means that every stage defers. Since $d_j\geq 1$,

$$\sum_{j=1}^{k}(|T_j|-d_j)\leq\sum_{j=1}^{k}(|T_j|-1)$$

Therefore, the MDT split penalty upper bounds the rule-list sparsity penalty of the corresponding rule-list representation.

F Experiment Setup

F.1 Machines

All experiments were performed on an institutional computing cluster. Each experiment was executed on a single compute node equipped with an AMD EPYC 9554 processor (2.75 GHz), with 64 physical cores. Resources were restricted to 1 CPU core and 50 GB of RAM.

F.2 Datasets

We provide a description of all datasets used in this work and the binary classification task we chose.

For all datasets, we remove rows containing missing values and one-hot encode all categorical variables.

Abalone [57] (*4,177 samples*) Predict whether an abalone is male based on its physical measurements. CC BY 4.0 License.

Adult [7] (*48,842 samples*) Predict whether an individual earns more than $50,000 per year based on demographic and occupational attributes. CC BY 4.0 License.

Aging [2, 44] (*714 samples*) Predict whether an individual has visited at least two doctors. CC BY 4.0 License.

Bank [51, 52] (*45,211 samples*) Predict whether a customer subscribes after being contacted. CC BY 4.0 License.

Bike [24, 25] (*17,379 samples*) Predict whether bike rental demand exceeds the median. CC BY 4.0 License.

California [58] (*20,634 samples*) Predict whether the provided binary target variable `price` is true. We believe this dataset is licensed under CC0, although the OpenML page does not explicitly specify a license. The dataset is derived from California census data.

Churn [23, 39] (*5,000 samples*) Predict whether a customer will churn. License unspecified; publicly available on OpenML and included in a recent benchmark [23].

Compas [6] (*4,966 samples*) Predict whether a defendant will recidivate within two years. Derived from anonymized public criminal records; available through the GitHub repository accompanying **?**], which does not specify a license (`https://github.com/propublica/compas-analysis`).

Coupon [1] (*108 samples*) Predict whether an individual accepts a recommended coupon. CC BY 4.0 License.

Credit [80, 81] (*30,000 samples*) Predict whether a client will default on their credit card payment in the following month. CC BY 4.0 License.

Diamonds [56] (*53,940 samples*) Predict whether a diamond belongs to cut category 2 based on its physical attributes. License unspecified on OpenML, although its linked GitLab mirror indicates a CC0 1.0 Universal license. The dataset is included in a standard benchmark [**?**].

Droid [45, 46] (*29,332 samples*) Predict whether an Android application is malicious. CC BY 4.0 License.

Heloc [27] (*2,502 samples*) Predict whether an individual is high- or low-risk for a home equity line of credit. The HELOC dataset is subject to a license permitting research use, originally documented at `https://community.fico.com/s/explainable-machine-learning-challenge?tabset-158d9=2`, although this page is no longer accessible.

Jasmine [31, 54] (*2,984 samples*) Binary classification using the provided target column. License unspecified on OpenML, although its linked GitLab mirror indicates a CC0 1.0 Universal license. The dataset was also featured in an open machine learning challenge [31].

Madeline [55] (*3,140 samples*) Binary classification using the provided target column. License unspecified on OpenML, although its linked GitLab mirror indicates a CC0 1.0 Universal license. The dataset was included in an AutoML Challenge [31].

Magic [10] (*19,020 samples*) Predict whether a Cherenkov telescope image corresponds to a gamma ray or background noise. CC BY 4.0 License.

Monk2 [73, 79] (*601 samples*) Predict the logical rule where the label is 1 if exactly two of six attributes take value 1. CC BY 4.0 License.

Phishing [49, 50] (*11,055 samples*) Predict whether a website is phishing or legitimate. CC BY 4.0 License.

Pol [59] (*10,082 samples*) Binary classification using the provided target column. License unspecified on OpenML, although its linked GitLab mirror indicates a CC0 1.0 Universal license. The dataset is included in a standard benchmark [**?**].

Rl [60] (*4,970 samples*) Binary classification using the provided target column. License unspecified on OpenML, although its linked GitLab mirror indicates a CC0 1.0 Universal license. The dataset is included in a standard benchmark [**?**].

Shopping [66, 67] (*12,330 samples*) Predict whether an online shopping session ends in a purchase. CC BY 4.0 License.

Spambase [34] (*4,601 samples*) Predict whether an email is spam. CC BY 4.0 License.

Student [18, 19] (*649 samples*) Predict whether a student passes a course (final grade ≥ 10). CC BY 4.0 License.

Tic-Tac-Toe [4] (*958 samples*) Predict whether a player has won given a terminal board configuration. CC BY 4.0 License.

Wine [61] (*6,497 samples*) Predict whether wine quality is at least 7. CC BY 4.0 License.

F.3 Hyperparameter Selection

All cross-validation was done with 3 folds. Any results reported are averaged across 5 train/test splits.

F.3.1 Black Boxes:

Random Forest. Random Forest hyperparameters were tuned with Optuna using a 12-hour timeout per train/test split:

$$\begin{aligned}
n_{\text{estimators}} &\in \{100, 200, \ldots, 1200\},\\
\text{max_depth} &\in [2, 32],\\
\text{min_samples_split} &\in [2, 40],\\
\text{min_samples_leaf} &\in [1, 20],\\
\text{max_features} &\in \{\sqrt{p}, \log_2(p), \alpha p : \alpha \in [0.1, 1.0]\},\\
\text{bootstrap} &\in \{\text{True}, \text{False}\},\\
\text{class_weight} &\in \{\text{None}, \text{balanced}, \text{balanced_subsample}\}.
\end{aligned}$$

XGBoost. XGBoost hyperparameters were tuned with Optuna using a 12-hour timeout per train/test split:

$$
\begin{aligned}
n_{\text{estimators}} &\in \{100, 200, \ldots, 1200\},\\
\text{max_depth} &\in [2, 12],\\
\text{learning_rate} &\in [10^{-3}, 3 \cdot 10^{-1}] \text{ on a log scale},\\
\text{min_child_weight} &\in [10^{-2}, 20] \text{ on a log scale},\\
\text{subsample} &\in [0.5, 1.0],\\
\text{colsample_bytree} &\in [0.5, 1.0],\\
\gamma &\in [10^{-8}, 10] \text{ on a log scale},\\
\alpha_{\text{reg}} &\in [10^{-8}, 10] \text{ on a log scale},\\
\lambda_{\text{reg}} &\in [10^{-8}, 10] \text{ on a log scale}.
\end{aligned}
$$

AdaBoost. AdaBoost [28] hyperparameters were tuned with Optuna using a 12-hour timeout per train/test split. We tuned th AdaBoost ensemble using the following search space:

$$
\begin{aligned}
n_{\text{estimators}} &\in \{50, 100, \ldots, 2000\},\\
\text{learning_rate} &\in [10^{-3}, 2] \text{ on a log scale},\\
\text{base_max_depth} &\in [1, 8],\\
\text{base_min_samples_split} &\in [2, 50],\\
\text{base_min_samples_leaf} &\in [1, 50],\\
\text{base_max_features} &\in \{\text{None}, \sqrt{p}, \log_2(p)\},\\
\text{base_criterion} &\in \{\text{gini}, \text{entropy}, \text{log_loss}\},
\end{aligned}
$$

Hyperparameters were selected to maximize mean cross-validation accuracy. The best configuration was retrained on the full training split and evaluated on the held-out test split.

F.3.2 Additive Models with Pairwise Interactions

Explainable Boosting Machine (EBM). EBM hyperparameters were tuned with Optuna using a 12-hour timeout per train/test split:

$$
\begin{aligned}
\text{interactions} &\in [0, 50],\\
\text{learning_rate} &\in [10^{-3}, 5 \cdot 10^{-2}] \text{ on a log scale},\\
\text{max_rounds} &\in \{200, 400, \ldots, 5000\},\\
\text{max_bins} &\in \{64, 128, 256, 512, 1024\},\\
\text{max_interaction_bins} &\in \{16, 32, 64, 128\},\\
\text{min_samples_leaf} &\in [2, 50],\\
\text{max_leaves} &\in [2, 5],\\
\text{outer_bags} &\in \{4, 8, 12, 16\},\\
\text{validation_size} &\in \{0.10, 0.15, 0.20\},\\
\text{early_stopping_rounds} &\in \{50, 100, 200\},\\
\text{early_stopping_tolerance} &\in [10^{-6}, 10^{-3}] \text{ on a log scale}.
\end{aligned}
$$

F.3.3 Sparse Ensembles

FIGS. FIGS hyperparameters were tuned with Optuna using a 12-hour timeout per train/test split. The following search space was used:

$$
\begin{aligned}
\text{max_rules} &\in [4, 64],\\
\text{max_trees} &\in [2, 20],\\
\text{max_depth} &\in [2, 12],\\
\text{max_features} &\in \{\text{None}, \sqrt{p}, \log_2(p)\},\\
\text{min_impurity_decrease} &\in [10^{-8}, 10^{-4}] \text{ on a log scale}.
\end{aligned}
$$

The interpretability constraints max_rules ≤ 64, max_trees ≤ 20, and max_depth ≤ 12 were always enforced.

F.3.4 Interpretable Models

SingleTree. For a single decision tree algorithm, we used our DeferTree algorithm with $\eta = 10^9$ to ensure no deferral (we also verified this on all results reported). This corresponds to a version of the near-optimal decision tree algorithm LicketySPLIT [5].

$$\lambda \in \{10^{-5}, 5 \cdot 10^{-5}, 10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}\},$$

We fixed a depth budget of 10, using λ to grow a sparser tree. This method requires a binarized dataset. We binarized the dataset with ThresholdGuessing [48], using $n_{\text{estimators}} = 150$ and max_depth $= 2$.

F.3.5 Hybrid Models

HyRS. For HyRS, we tuned

$$\begin{aligned}
\text{nrules} &\in \{5000\},\\
\text{supp} &\in \{5, 20\},\\
\text{maxlen} &\in \{2, 3\},\\
\alpha &\in \{10^{-4}, 10^{-3}, 10^{-2}\},\\
\beta &\in \{0.01, 0.05, 0.10, 0.15, 0.25, 0.50\},\\
\text{niter} &\in \{5000\}.
\end{aligned}$$

Non-binary numeric features were binarized using 10 quantile levels. We used their provided Random Forest rule mining approach.

FLMM. For FLMM, we tuned

$$\begin{aligned}
r_{\max} &\in \{1, 2, 3\},\\
\ell &\in \{0.01, 0.02, 0.05, 0.1, 0.2\},\\
s &\in \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\},\\
\text{min_support_frac} &\in \{0.005, 0.01, 0.02\}.
\end{aligned}$$

We could not find an exact description of the rule mining done in [30], except that $r_{\max}$ was tuned in $\{1, 2, 3\}$. Therefore, we use the following procedure.

Non-binary numeric features were binarized using 10 quantile levels. Then, candidate rules were exhaustively enumerated as all conjunctions of between 1 and $r_{\max}$ binary indicators. Any rule whose support was below

$$\max(1, \lceil \text{min_support_frac} \cdot n \rceil)$$

was discarded. For each remaining conjunction, the predicted label was chosen as the majority class among covered samples. Duplicate rules inducing identical coverage and prediction behavior were removed.

HybridCORELS-POST. For the HybridCORELS-POST variant, we tuned

$$\begin{aligned}
c &\in \{0.001, 0.01, 0.1\},\\
\text{min_coverage} &\in \{0.1, 0.25, 0.5, 0.75, 0.9\},\\
\text{max_card} &\in \{1, 2\},\\
\text{min_support} &\in \{1, 5\},\\
\text{n_rules} &\in \{300\},
\end{aligned}$$

HybridCORELS-PRE. For the HybridCORELS-PRE variant, we tuned

$$\begin{aligned} c &\in \{0.001, 0.01, 0.1\}, \\ \text{min_coverage} &\in \{0.1, 0.25, 0.5, 0.75, 0.9\}, \\ \text{max_card} &\in \{1, 2\}, \\ \text{min_support} &\in \{5\}, \\ \text{n_rules} &\in \{300\}, \\ \alpha &\in \{0, 5, 10\}. \end{aligned}$$

For both optimal methods (HybridCORELS PRE and POST), we set a time limit of 10 minutes per hyperparameter to fit the rule list. This does not include training the black box or performing rule mining.

For the HybridCORELS variants, we first binarized the dataset with ThresholdGuessing [48], using $n_{\text{estimators}} = 150$ and max_depth $= 2$. We then used their provided rule mining method on these binary features, which uses FPGrowth. Because the HybridCORELS implementation exposes prediction through a `predict` method that accepts only the already rule-mined dataset, the fallback XGBoost model was required to operate on this same mined feature space. As such, we retuned the XGBoost model for each hyperparameter combination for HybridCORELS, because the HybridCORELS hyperparameters can change the rule mining and thus what XGBoost is trained on. Aside from that, the method of choosing hyperparameters is discussed in subsection F.4.

DeferTree. For our DeferTree, we tuned

$$\begin{aligned} \lambda &\in \{10^{-5}, 5 \cdot 10^{-5}, 10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}\}, \\ \eta &\in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.3, 0.6, 0.8, 1.0, 10^{9}\}. \end{aligned}$$

This method requires a binarized dataset. We binarized the dataset with ThresholdGuessing [48], using $n_{\text{estimators}} = 150$ and max_depth $= 2$.

MDT. For our MDT, we tuned

$$\begin{aligned} \lambda &\in \{10^{-5}, 5 \cdot 10^{-5}, 10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}\}, \\ \eta &\in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.3, 0.6, 0.8, 1.0\}, \\ \text{rescale_tau} &\in \{0, 1\}, \\ c &\in \{0, 0.2, 0.5, 1.0\}, \\ \gamma &\in \{0, 1, 2, 4, 8\}, \end{aligned}$$

The fixed settings were depth budget 10, maximum stages 6, maximum total stage leaves 500, maximum leaves in an uncompressed single tree 10^6, and maximum leaves per tree 129. In practice, we do not hit depth 10 or anywhere near these limits (shown in Figure 4a); these are upper bounds that would only prune very undesired hyperparameters. In our algorithm, the calls to fit a DeferTree require a binarized dataset. We binarized the dataset with ThresholdGuessing [48], using $n_{\text{estimators}} = 150$ and max_depth $= 2$.

F.4 Cross Validation Procedure

For hybrid interpretable models with fallback black boxes, hyperparameter selection was performed separately for each dataset train/test split. For a given train/test split, we constructed a table whose rows corresponded to hybrid-model hyperparameter combinations and whose columns stored the average validation accuracy, average validation deferral rate, test accuracy, and test deferral rate.

To populate this table, we first created three subtrain/validation partitions from the training set. For each subtrain partition, we independently tuned an XGBoost model over the ranges described earlier using an inner 3-fold cross-validation procedure restricted to that subtrain partition. After selecting the best XGBoost hyperparameters for that subtrain partition, we fixed that XGBoost model and evaluated every hyperparameter combination in the hybrid interpretable grid using the associated validation split. Repeating this process across the three subtrain/validation partitions yielded an

average validation accuracy and average validation deferral rate for each hybrid hyperparameter configuration.

Next, we tuned XGBoost once more on the entire training set using the same XGBoost search procedure. The resulting XGBoost model was then used as the fallback model for every hybrid hyperparameter configuration trained on the full training set, producing a corresponding test accuracy and test deferral rate.

Given this matrix for a dataset train/test split, if we wished to select the best model subject to at most $x\%$ test deferral, we first filtered out all hyperparameter configurations whose test deferral exceeded the threshold. Among the remaining configurations, we selected the one with the highest average validation accuracy. We then reported the corresponding test accuracy and test deferral for that configuration. We note that the former (the test accuracy) had not been looked at before then, and was not used to choose any hyperparameters. This entire procedure was repeated independently across the five train/test splits for each dataset.

G Experimental Results

Dataset	Defer Tree Non-Deferred Error	Defer Tree Deferred Error
Bike	$1.57\% \pm 0.42\%$	$14.88\% \pm 2.35\%$
Churn	$3.04\% \pm 0.33\%$	$14.16\% \pm 4.21\%$
Compas	$31.59\% \pm 2.11\%$	$36.24\% \pm 1.08\%$
Heloc	$25.59\% \pm 2.61\%$	$36.43\% \pm 6.78\%$
Jasmine	$13.97\% \pm 2.46\%$	$32.39\% \pm 4.45\%$
Phishing	$0.68\% \pm 0.24\%$	$11.84\% \pm 1.54\%$
Pol	$0.07\% \pm 0.08\%$	$6.56\% \pm 0.81\%$
RL	$16.57\% \pm 2.30\%$	$24.34\% \pm 2.99\%$
Wine	$4.67\% \pm 0.88\%$	$19.49\% \pm 1.51\%$

Table 2: Test error comparison between the DeferTree non-deferred and deferred regions. The deferred regions consistently exhibit substantially higher test error, motivating that our recursive approach with adaptive weights could reduce the amount we rely on the black box with little change in accuracy. All values are mean $\pm$ standard deviation across 5 splits.

In Figure 5, we compare the semi-supervised hyperparameter selection procedure to a variant that uses training deferral rate as a proxy for test deferral rate. We utilized this semi-supervised procedure for all methods, and here we compare this approach to using the train deferral rate for MDT+XGB. For each target deferral level x, we choose the hyperparameters with the highest average validation accuracy among those with deferral rate at most x, where deferral is measured on the test set for MDT+XGB and on the training set for MDT+XGB Train. After selecting hyperparameters separately for each of the five train/test splits, we plot the average test accuracy against the average test deferral rate of the selected models. Across datasets, the two curves are nearly indistinguishable: using training deferral is sometimes slightly better and sometimes slightly worse, but it usually selects models with essentially the same test accuracy–deferral tradeoff. The only clear degradation occurs on Coupon, which has only 108 total samples after removing missing values, so little weight should be placed on that outlier. Overall, these results suggest that training deferral rate is an acceptable practical proxy for test deferral rate when selecting MDT+XGB hyperparameters.

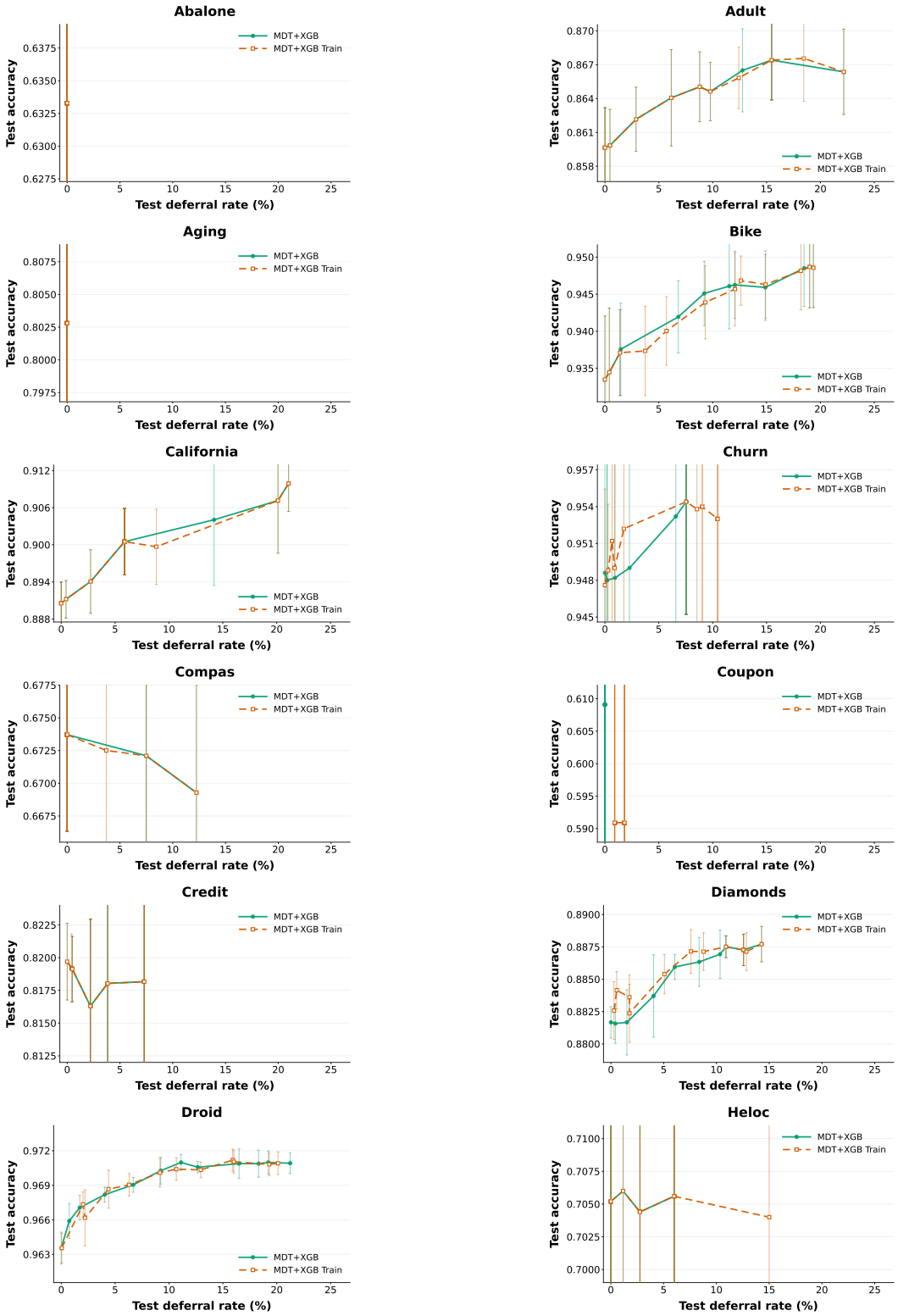


Figure 5: Comparison between MDT+XGB hyperparameter selection using test deferral rate and selection using training deferral rate.

Dataset	Difference in Train/Test Defer Rates	Train Defer Rate	Test Defer Rate
Abalone	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Adult	0.0052 ± 0.0046	0.2176 ± 0.0307	0.2216 ± 0.0309
Aging	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Bike	0.0055 ± 0.0047	0.1595 ± 0.0399	0.1539 ± 0.0397
California	0.0105 ± 0.0073	0.2208 ± 0.0142	0.2110 ± 0.0218
Churn	0.0037 ± 0.0040	0.0508 ± 0.0556	0.0532 ± 0.0544
Compas	0.0084 ± 0.0102	0.1240 ± 0.1148	0.1225 ± 0.1137
Coupon	0.0142 ± 0.0317	0.0233 ± 0.0520	0.0091 ± 0.0203
Credit	0.0043 ± 0.0029	0.0731 ± 0.0689	0.0730 ± 0.0721
Diamonds	0.0043 ± 0.0029	0.1658 ± 0.0450	0.1624 ± 0.0479
Droid	0.0033 ± 0.0021	0.1841 ± 0.0621	0.1841 ± 0.0648
Heloc	0.0105 ± 0.0118	0.0495 ± 0.0555	0.0600 ± 0.0671
Jasmine	0.0067 ± 0.0072	0.0613 ± 0.0729	0.0593 ± 0.0782
Madeline	0.0139 ± 0.0105	0.1150 ± 0.0843	0.1102 ± 0.0811
Magic	0.0027 ± 0.0023	0.2299 ± 0.0129	0.2301 ± 0.0128
Monk2	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Phishing	0.0109 ± 0.0061	0.1807 ± 0.0300	0.1873 ± 0.0376
Pol	0.0033 ± 0.0036	0.1474 ± 0.0497	0.1463 ± 0.0524
Rl	0.0094 ± 0.0079	0.1520 ± 0.0609	0.1441 ± 0.0617
Shopping	0.0057 ± 0.0041	0.2056 ± 0.0277	0.2015 ± 0.0250
Spambase	0.0146 ± 0.0124	0.1717 ± 0.0349	0.1863 ± 0.0301
Student	0.0084 ± 0.0157	0.0470 ± 0.0880	0.0554 ± 0.1037
Tictactoe	0.0047 ± 0.0065	0.0350 ± 0.0346	0.0323 ± 0.0358
Wine	0.0124 ± 0.0102	0.2101 ± 0.0539	0.1982 ± 0.0465

Table 3: Mean and standard deviation across train/test splits of the absolute difference between train and test defer rates, along with the train and test defer rates themselves, for MDT+XGB models selected under a 25% test defer-rate cutoff.

Dataset	Split Compression	Leaf Compression	MDT Deferred Accuracy	XGB Deferred Accuracy	Δ Accuracy (MDT $-$ XGB)
Adult	1.0214 ± 0.0167	1.0819 ± 0.0556	$0.7049\ [0.6959, 0.7139]$	$0.7067\ [0.6977, 0.7156]$	$-0.0018\ [-0.0048, 0.0012]$
Bike	1.0642 ± 0.0277	1.0875 ± 0.0803	$0.8082\ [0.7933, 0.8228]$	$0.8090\ [0.7940, 0.8236]$	$-0.0007\ [-0.0060, 0.0045]$
Compas	1.7564 ± 0.1381	1.5450 ± 0.6991	$0.6349\ [0.5970, 0.6727]$	$0.6332\ [0.5954, 0.6711]$	$0.0016\ [-0.0132, 0.0164]$
Diamonds	1.1116 ± 0.1536	1.4575 ± 0.5455	$0.7221\ [0.7126, 0.7314]$	$0.7252\ [0.7158, 0.7345]$	$-0.0031\ [-0.0066, 0.0003]$
Jasmine	1.8466 ± 0.5427	1.3607 ± 0.8066	$0.6723\ [0.6045, 0.7401]$	$0.6893\ [0.6215, 0.7571]$	$-0.0169\ [-0.0734, 0.0395]$
Madeline	1.0614 ± 0.0752	11.9562 ± 24.2114	$0.6994\ [0.6503, 0.7486]$	$0.7139\ [0.6647, 0.7601]$	$-0.0145\ [-0.0491, 0.0202]$
Phishing	1.8082 ± 0.3270	1.0475 ± 0.0443	$0.8798\ [0.8658, 0.8938]$	$0.8798\ [0.8658, 0.8938]$	$0.0000\ [-0.0029, 0.0024]$
Pol	1.0024 ± 0.0221	1.0434 ± 0.0628	$0.9186\ [0.9044, 0.9322]$	$0.9173\ [0.9031, 0.9308]$	$0.0014\ [-0.0047, 0.0075]$
Rl	1.2228 ± 0.0952	1.3269 ± 0.2278	$0.7458\ [0.7137, 0.7779]$	$0.7416\ [0.7095, 0.7737]$	$0.0042\ [-0.0112, 0.0196]$
Shopping	1.1049 ± 0.0796	1.3019 ± 0.3199	$0.6973\ [0.6787, 0.7150]$	$0.7009\ [0.6828, 0.7186]$	$-0.0036\ [-0.0121, 0.0052]$

Table 4: Compression ratios (mean $\pm$ standard deviation across splits) for splits to evaluate a deferred test sample and leaves of the final compressed MDT XGBoost relative to the original XGBoost fallback. A 95% confidence interval on the difference (MDT $-$ XGB) in test deferred region accuracy shows they are indistinguishable. In some settings, the MDT's XGBoost has higher mean accuracy. In other cases, it has lower mean accuracy. However, the confidence intervals almost always contain 0. This shows that we can obtain meaningful compression on the black box without any loss in accuracy. Hyperparameters were selected according to maximum validation accuracy without exceeding 25% test deferral for a train-test split. To compute confidence intervals, we concatenate the deferred-sample correctness indicators across all train/test splits and apply bootstrap resampling to the pooled vector of deferred samples.

Table 5: Runtime statistics of the chosen hyperparameters (best validation accuracy with under 25% test deferral) across datasets. Mean $\pm$ standard deviation across 5 train/test splits is shown.

Dataset	Mean $\pm$ Std (s)
Abalone	27.99 ± 2.44
Adult	367.56 ± 106.81
Aging	0.65 ± 0.12
Bike	102.39 ± 48.71
California	315.83 ± 28.55
Churn	57.19 ± 20.99
Compas	9.08 ± 2.34
Coupon	0.50 ± 0.13
Credit	459.45 ± 82.95
Diamonds	364.63 ± 185.63
Droid	53.54 ± 29.69
Heloc	25.59 ± 2.00
Jasmine	44.13 ± 16.54
Madeline	323.47 ± 126.18
Magic	427.09 ± 15.86
Monk2	0.52 ± 0.04
Phishing	18.49 ± 12.89
Pol	159.86 ± 37.14
Rl	37.65 ± 43.64
Shopping	146.12 ± 11.98
Spambase	160.96 ± 33.55
Student	2.08 ± 0.43
Tictactoe	2.22 ± 0.07
Wine	63.69 ± 5.84

Dataset	Difference in Train/Test Defer Rates	Train Defer Rate	Test Defer Rate
Abalone	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Adult	0.0017 ± 0.0016	0.0890 ± 0.0097	0.0879 ± 0.0099
Aging	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Bike	0.0060 ± 0.0050	0.0851 ± 0.0300	0.0791 ± 0.0295
California	0.0022 ± 0.0016	0.0608 ± 0.0070	0.0588 ± 0.0059
Churn	0.0053 ± 0.0044	0.0437 ± 0.0414	0.0444 ± 0.0380
Compas	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Coupon	0.0142 ± 0.0317	0.0233 ± 0.0520	0.0091 ± 0.0203
Credit	0.0021 ± 0.0019	0.0234 ± 0.0139	0.0220 ± 0.0145
Diamonds	0.0030 ± 0.0021	0.0840 ± 0.0186	0.0810 ± 0.0180
Droid	0.0048 ± 0.0040	0.0873 ± 0.0127	0.0832 ± 0.0103
Heloc	0.0048 ± 0.0066	0.0228 ± 0.0322	0.0276 ± 0.0386
Jasmine	0.0044 ± 0.0071	0.0258 ± 0.0360	0.0214 ± 0.0312
Madeline	0.0094 ± 0.0068	0.0482 ± 0.0289	0.0576 ± 0.0332
Magic	0.0072 ± 0.0053	0.0664 ± 0.0298	0.0669 ± 0.0304
Monk2	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Phishing	0.0048 ± 0.0042	0.0914 ± 0.0039	0.0903 ± 0.0047
Pol	0.0059 ± 0.0021	0.0728 ± 0.0242	0.0719 ± 0.0188
Rl	0.0046 ± 0.0047	0.0436 ± 0.0463	0.0390 ± 0.0416
Shopping	0.0062 ± 0.0046	0.0691 ± 0.0336	0.0648 ± 0.0289
Spambase	0.0046 ± 0.0027	0.0702 ± 0.0220	0.0739 ± 0.0194
Student	0.0011 ± 0.0026	0.0066 ± 0.0146	0.0077 ± 0.0172
Tictactoe	0.0047 ± 0.0065	0.0350 ± 0.0346	0.0323 ± 0.0358
Wine	0.0069 ± 0.0077	0.0527 ± 0.0412	0.0594 ± 0.0468

Table 6: Mean and standard deviation across train/test splits of the absolute difference between train and test defer rates, along with the train and test defer rates themselves, for MDT+XGB models selected under a 10% test defer-rate cutoff.

Dataset	Non-Deferral Tree	MDT+XGB $\leq$25%	FIGS	Random Forest	XGBoost
Abalone	0.6314 ± 0.0135	0.6333 ± 0.0155	0.6381 ± 0.0066	0.6314 ± 0.0079	0.6323 ± 0.0157
Adult	0.8598 ± 0.0038	0.8667 ± 0.0047	0.8589 ± 0.0048	0.8619 ± 0.0038	0.8701 ± 0.0030
Aging	0.8028 ± 0.0077	0.8028 ± 0.0077	0.7944 ± 0.0106	0.8014 ± 0.0080	0.8000 ± 0.0063
Bank	0.9051 ± 0.0008	0.9052 ± 0.0017	0.9069 ± 0.0008	$\underline{0.9087 \pm 0.0008}$	$\mathbf{0.9090 \pm 0.0007}$
Bike	0.9346 ± 0.0046	0.9486 ± 0.0054	0.9113 ± 0.0087	0.9426 ± 0.0037	0.9500 ± 0.0052
California	0.8893 ± 0.0053	0.9099 ± 0.0045	0.8774 ± 0.0034	0.9019 ± 0.0072	0.9101 ± 0.0050
Churn	0.9424 ± 0.0081	0.9566 ± 0.0063	0.9434 ± 0.0127	0.9506 ± 0.0072	0.9574 ± 0.0063
Compas	0.6725 ± 0.0076	0.6693 ± 0.0083	0.6657 ± 0.0112	0.6741 ± 0.0121	0.6757 ± 0.0072
Coupon	0.6091 ± 0.0689	0.6000 ± 0.0674	0.5727 ± 0.0407	0.5364 ± 0.0498	0.5545 ± 0.0674
Credit	0.8196 ± 0.0032	0.8182 ± 0.0065	0.8200 ± 0.0032	0.8195 ± 0.0029	0.8202 ± 0.0029
Diamonds	0.8815 ± 0.0016	0.8877 ± 0.0014	0.8829 ± 0.0023	0.8871 ± 0.0009	0.8883 ± 0.0021
Droid	0.9627 ± 0.0024	0.9709 ± 0.0012	0.9601 ± 0.0011	0.9715 ± 0.0014	0.9718 ± 0.0009
Heloc	0.7052 ± 0.0239	0.7056 ± 0.0239	0.6840 ± 0.0165	0.7080 ± 0.0228	0.7116 ± 0.0241
Jasmine	0.8037 ± 0.0173	0.8161 ± 0.0224	0.7983 ± 0.0120	0.8218 ± 0.0190	0.8214 ± 0.0158
Madeline	0.8408 ± 0.0281	0.8627 ± 0.0122	0.8099 ± 0.0328	0.8793 ± 0.0158	0.8697 ± 0.0115
Magic	0.8541 ± 0.0066	0.8756 ± 0.0111	0.8539 ± 0.0045	0.8798 ± 0.0059	0.8849 ± 0.0064
Monk2	1.0000 ± 0.0000	1.0000 ± 0.0000	0.8867 ± 0.0402	0.9933 ± 0.0149	1.0000 ± 0.0000
Phishing	0.9528 ± 0.0039	0.9728 ± 0.0038	0.9392 ± 0.0042	0.9721 ± 0.0029	0.9733 ± 0.0034
Pol	0.9723 ± 0.0026	0.9849 ± 0.0006	0.9631 ± 0.0040	0.9823 ± 0.0016	0.9845 ± 0.0015
Rl	0.7614 ± 0.0117	0.7968 ± 0.0198	0.7163 ± 0.0210	0.8074 ± 0.0129	0.8278 ± 0.0075
Shopping	0.9010 ± 0.0055	0.9047 ± 0.0036	0.8988 ± 0.0048	0.9046 ± 0.0027	0.9057 ± 0.0034
Spambase	0.9254 ± 0.0035	0.9500 ± 0.0053	0.9222 ± 0.0077	0.9517 ± 0.0031	0.9528 ± 0.0051
Student	0.8292 ± 0.0305	0.8154 ± 0.0361	0.8046 ± 0.0241	0.8215 ± 0.0358	0.8046 ± 0.0301
Tictactoe	0.9375 ± 0.0165	0.9885 ± 0.0113	0.9552 ± 0.0291	0.9896 ± 0.0037	1.0000 ± 0.0000
Wine	0.8340 ± 0.0033	0.8788 ± 0.0116	0.8391 ± 0.0087	0.8848 ± 0.0095	0.8864 ± 0.0049

Table 7: Test accuracy (mean $\pm$ standard deviation) across all datasets.

Table 8: Black-box evaluation complexity at a 25% deferral rate cap, for the best-performing MDT at no more than that test deferral rate. For each deferred test example, we count the split decisions traversed across all XGBoost trees, then average over deferred examples from the completed train/test splits. *Root XGBoost* uses the root black-box model as the fallback without retraining. *Retrained XGBoost* retrains the black box at each stage as in our algorithm, without compression. *Compressed single XGBoost* applies compression using the union of deferred subregions. *Subregion-specific XGBoost* compresses a model separately within each deferred subregion rather than over their union. Lower is better.

Dataset	Subregion-specific XGBoost	Compressed single XGBoost	Retrained XGBoost	Root XGBoost
Abalone	1,863	3,546	3,569	3,579
Adult	4,007	5,442	5,514	5,516
Bike	3,241	5,319	5,319	5,549
Churn	3,428	4,115	4,116	4,903
Compas	798	798	1,030	1,049
Coupon	2,266	2,266	2,689	2,662
Diamonds	5,268	7,193	7,508	7,848
Droid	8,042	9,287	9,287	9,009
Heloc	899	953	967	971
Jasmine	2,480	2,533	2,533	3,969
Madeline	3,896	4,366	4,366	4,593
Phishing	1,985	2,550	2,611	4,071
Pol	3,106	4,047	4,078	4,122
Rl	3,046	4,458	4,634	6,265
Shopping	2,540	2,579	2,616	2,963
Spambase	3,190	3,937	3,943	4,095
Tictactoe	2,240	2,656	2,656	2,834
Wine	4,076	6,862	6,862	7,004

Table 9: Normalized black-box evaluation complexity at a 25% target deferral cap, for the best-performing MDT at that deferral rate. For each deferred test example, we count the split decisions traversed across all XGBoost trees, then average over deferred examples from the completed train/test splits. *Root XGBoost* uses the root black-box model as the fallback without retraining. *Retrained XGBoost* retrains the black box at each stage as in our algorithm, without compression. *Compressed single XGBoost* applies compression using the union of deferred subregions. *Subregion-specific XGBoost* compresses a model separately within each deferred subregion rather than over their union. Every value is divided by the corresponding Root XGBoost count; lower is better.

Dataset	Subregion-specific XGBoost	Compressed single XGBoost	Retrained XGBoost	Root XGBoost
abalone	0.521	0.991	0.997	1.000
adult	0.726	0.987	1.000	1.000
bike	0.584	0.959	0.959	1.000
churn	0.699	0.839	0.839	1.000
compas	0.761	0.761	0.982	1.000
coupon	0.851	0.851	1.010	1.000
diamonds	0.671	0.917	0.957	1.000
droid	0.893	1.031	1.031	1.000
heloc	0.926	0.981	0.996	1.000
jasmine	0.625	0.638	0.638	1.000
madeline	0.848	0.951	0.951	1.000
phishing	0.488	0.626	0.641	1.000
pol	0.754	0.982	0.989	1.000
rl	0.486	0.712	0.740	1.000
shopping	0.857	0.870	0.883	1.000
spambase	0.779	0.961	0.963	1.000
tictactoe	0.790	0.937	0.937	1.000
wine	0.582	0.980	0.980	1.000

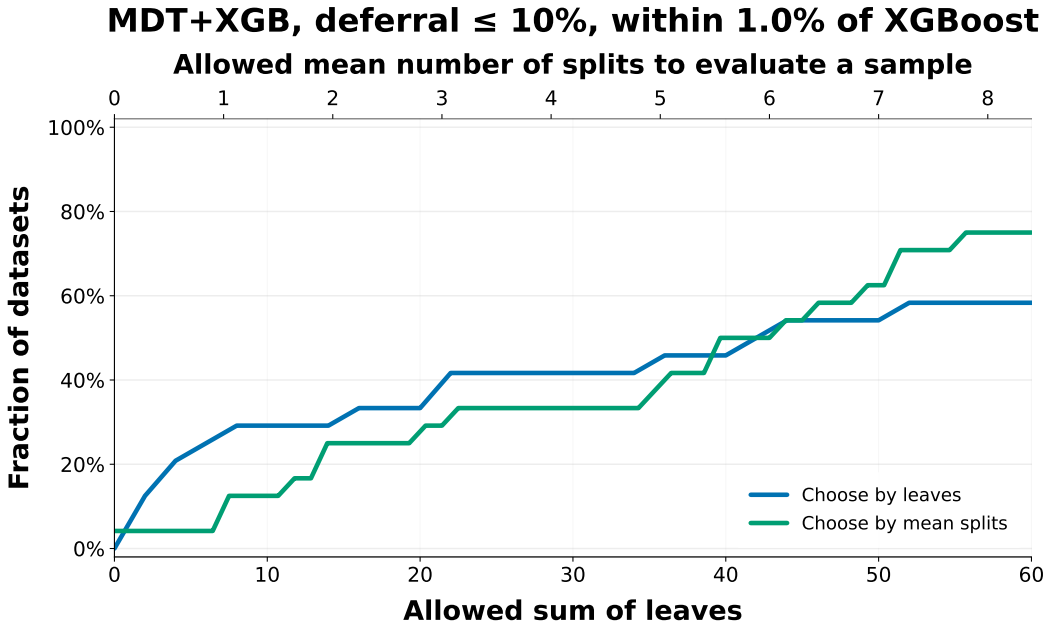


Figure 6: Comparison between selecting hyperparameters for MDTs using a budget on the total number of leaves versus a budget on the mean number of split decisions required to evaluate a sample. The curves are closely but not exactly aligned. A rule of thumb is that a total of 50 leaves corresponds to a mean of approximately 7 split evaluations per sample. This is much better than a full depth 7 tree, which has 128 leaves.



Figure 7: Legend for the accuracy-deferral trade-off plots on the following pages.

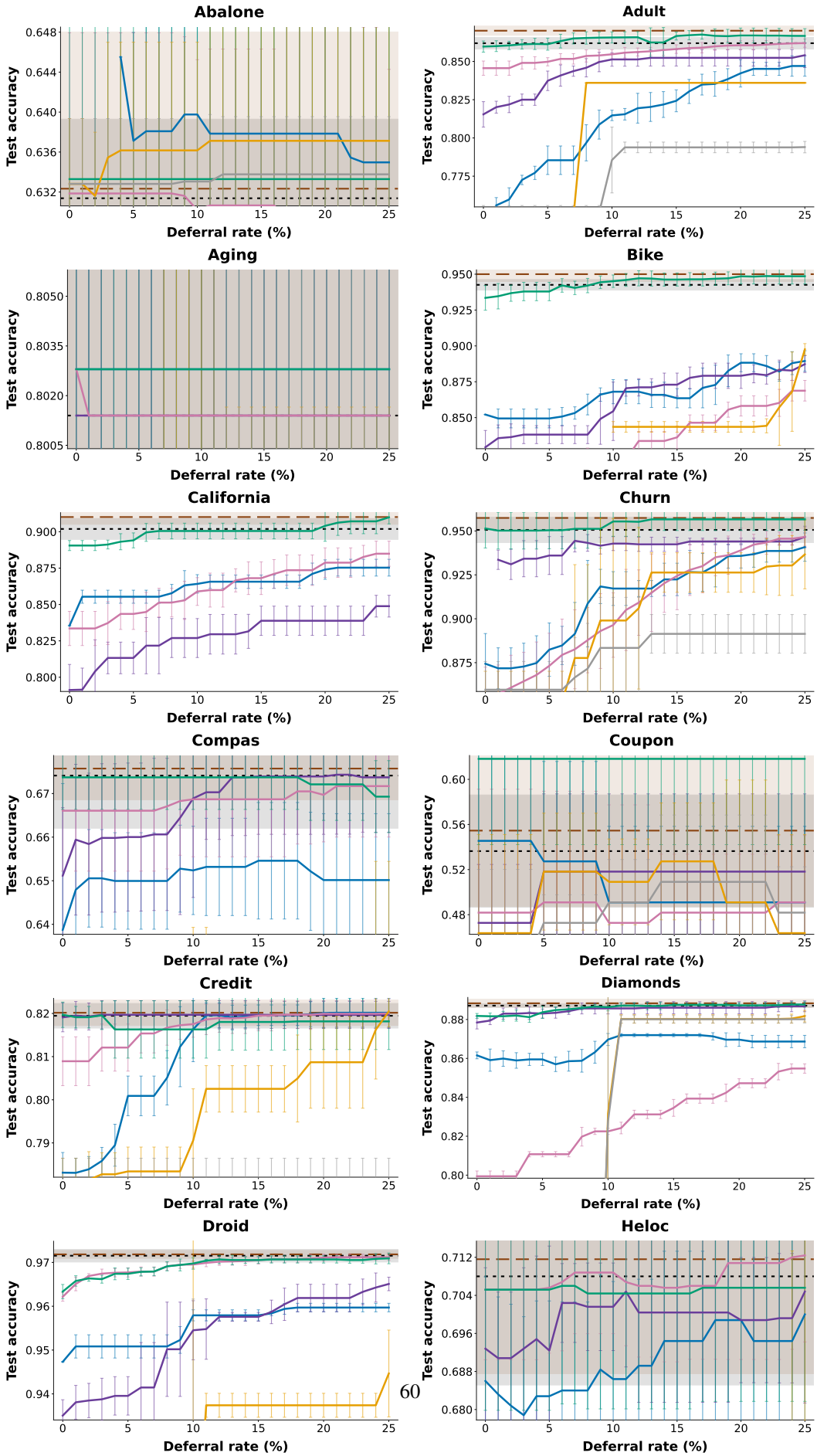


Figure 8: Deferral-accuracy trade-offs for all datasets.

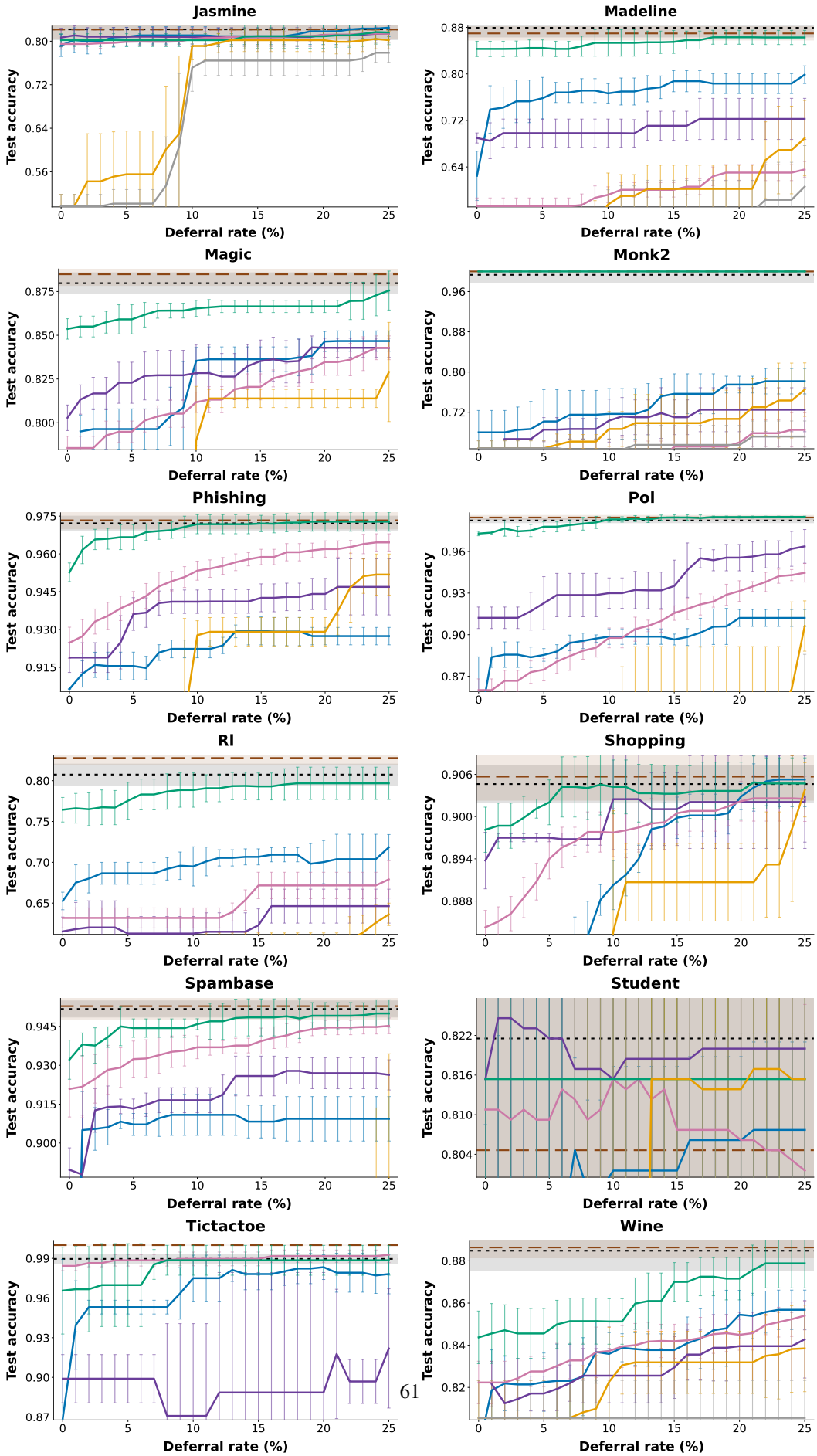


Figure 9: Deferral-accuracy trade-offs for all datasets, continued.

Table 10: Comparison of multistage defer trees with XGBoost fallback (MDT+XGB), constrained to at most 25% deferral, against AdaBoost. Entries report mean test classification accuracy $\pm$ standard error over five train/test splits (higher is better).

Dataset	MDT+XGB ($\leq$25%)	AdaBoost
Abalone	0.6333 ± 0.0069	0.6326 ± 0.0026
Adult	0.8667 ± 0.0021	0.8684 ± 0.0016
Aging	0.8028 ± 0.0034	0.8000 ± 0.0028
Bank	0.9087 ± 0.0008	0.9092 ± 0.0009
Bike	0.9486 ± 0.0024	0.9461 ± 0.0020
California	0.9009 ± 0.0020	0.9088 ± 0.0018
Churn	0.9566 ± 0.0028	0.9548 ± 0.0021
Droid	0.9709 ± 0.0005	0.9711 ± 0.0004
Heloc	0.7056 ± 0.0107	0.7124 ± 0.0115
Jasmine	0.8161 ± 0.0100	0.8241 ± 0.0089
Phishing	0.9728 ± 0.0017	0.9731 ± 0.0016
Pol	0.9849 ± 0.0003	0.9843 ± 0.0010
RL	0.7968 ± 0.0089	0.8052 ± 0.0071
Shopping	0.9047 ± 0.0016	0.9062 ± 0.0017
Spambase	0.9500 ± 0.0024	0.9524 ± 0.0024
Wine	0.8788 ± 0.0052	0.8870 ± 0.0035

G.1 Ablations

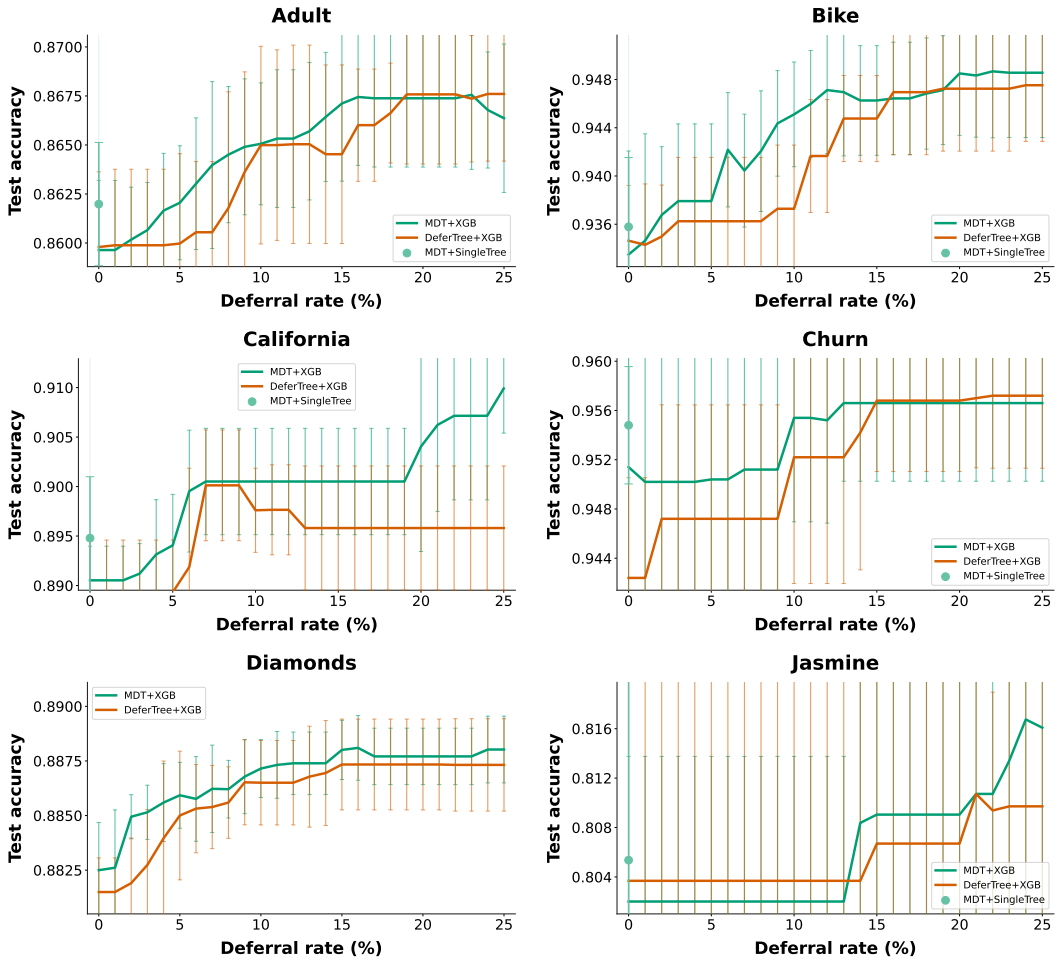


Figure 10: Results of our MDT algorithm compared to just using our DeferTree algorithm. Here, we also show results for MDT+SingleTree, which uses a single decision tree algorithm as fallback instead of XGBoost. When the deferral rate is 0, DeferTree+XGB turns into a near-optimal decision tree algorithm. We see that MDT+SingleTree is able to provide a much better, fully interpretable model than a near-optimal tree in most cases.

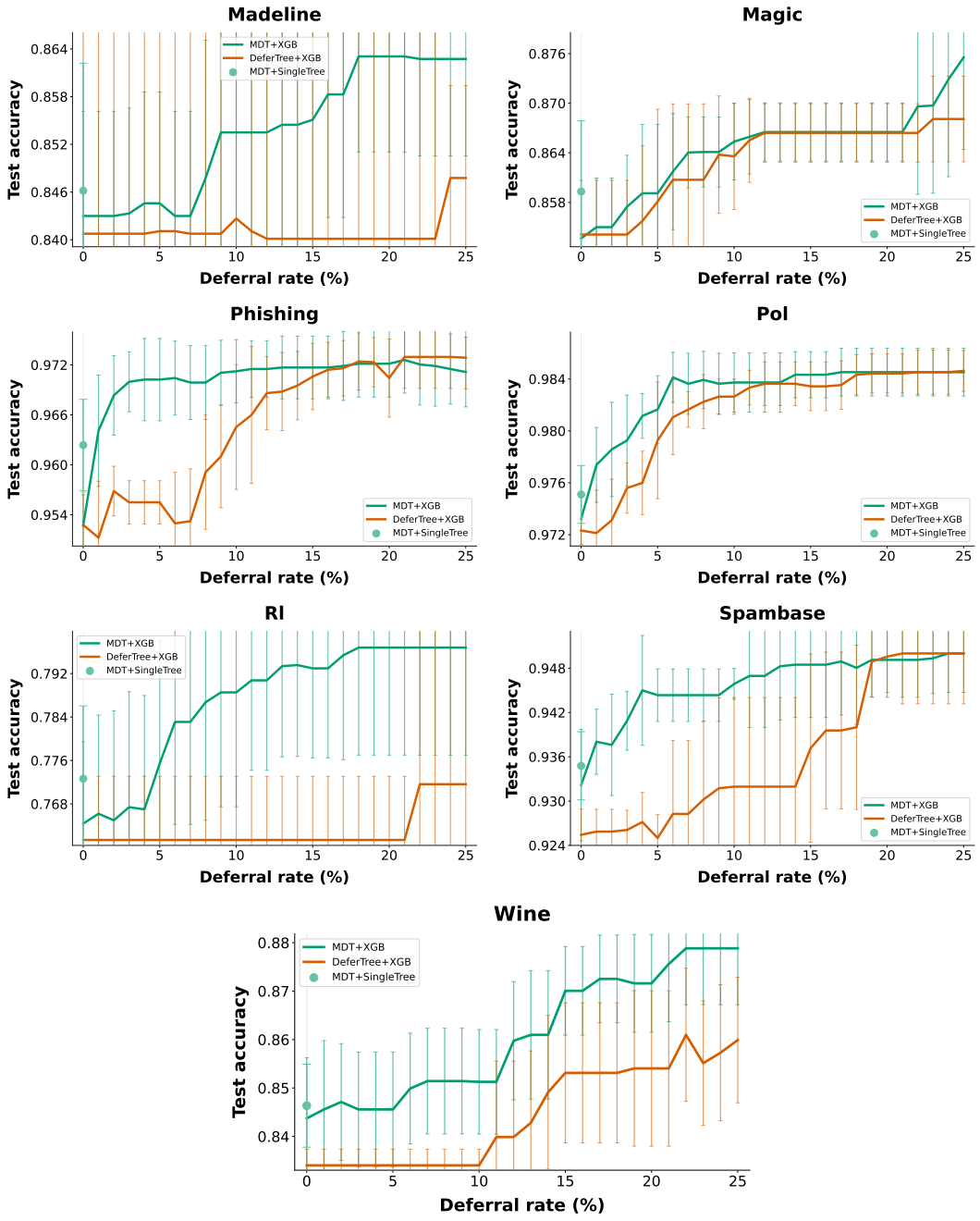


Figure 11: Results of our MDT algorithm compared to just using our DeferTree algorithm (continued).

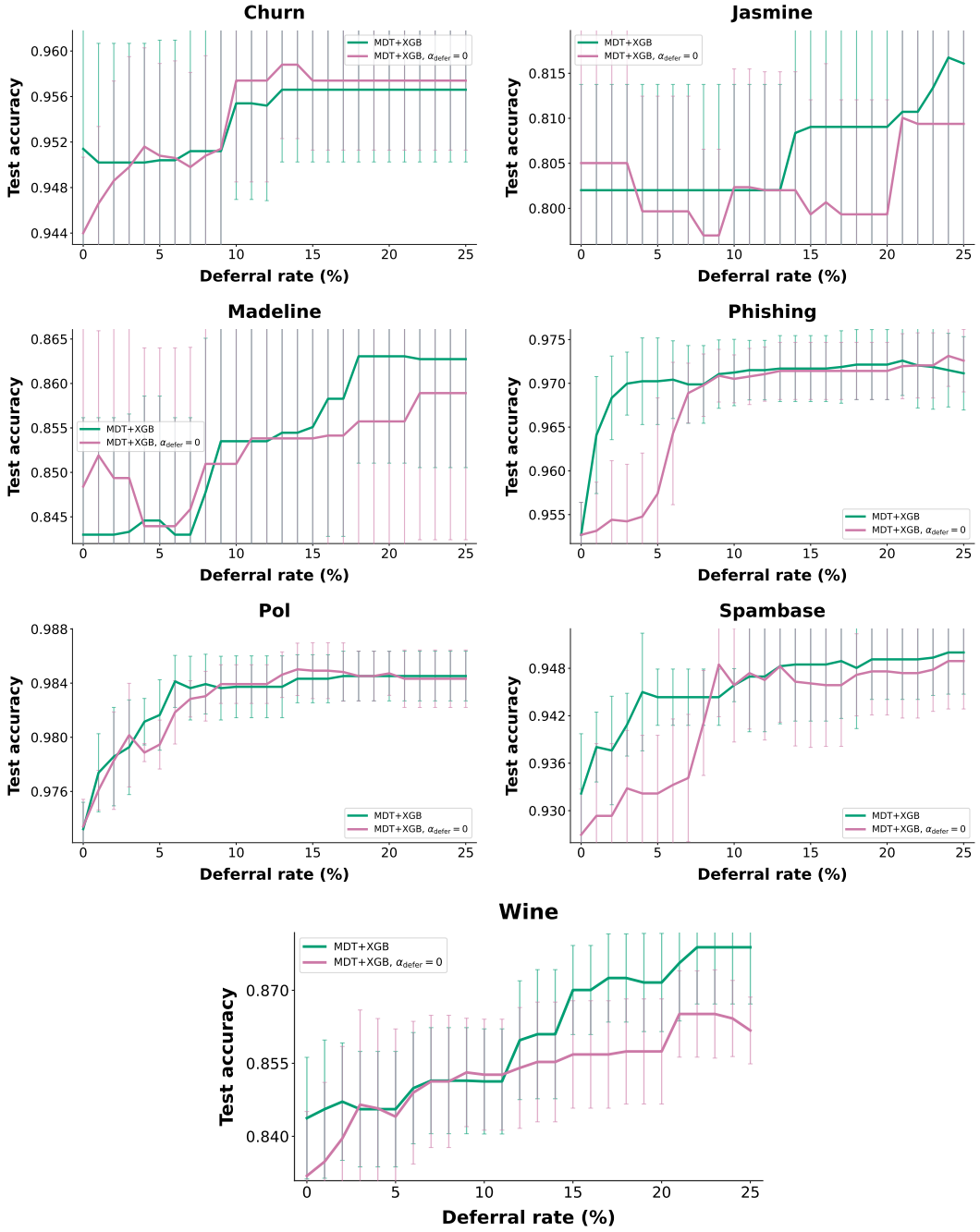


Figure 12: Ablation on the impacts of rescaling τ for later stages of the MDT. The boolean `rescale_tau` is set to 0 for the $\alpha_{\text{defer}} = 0$ method.

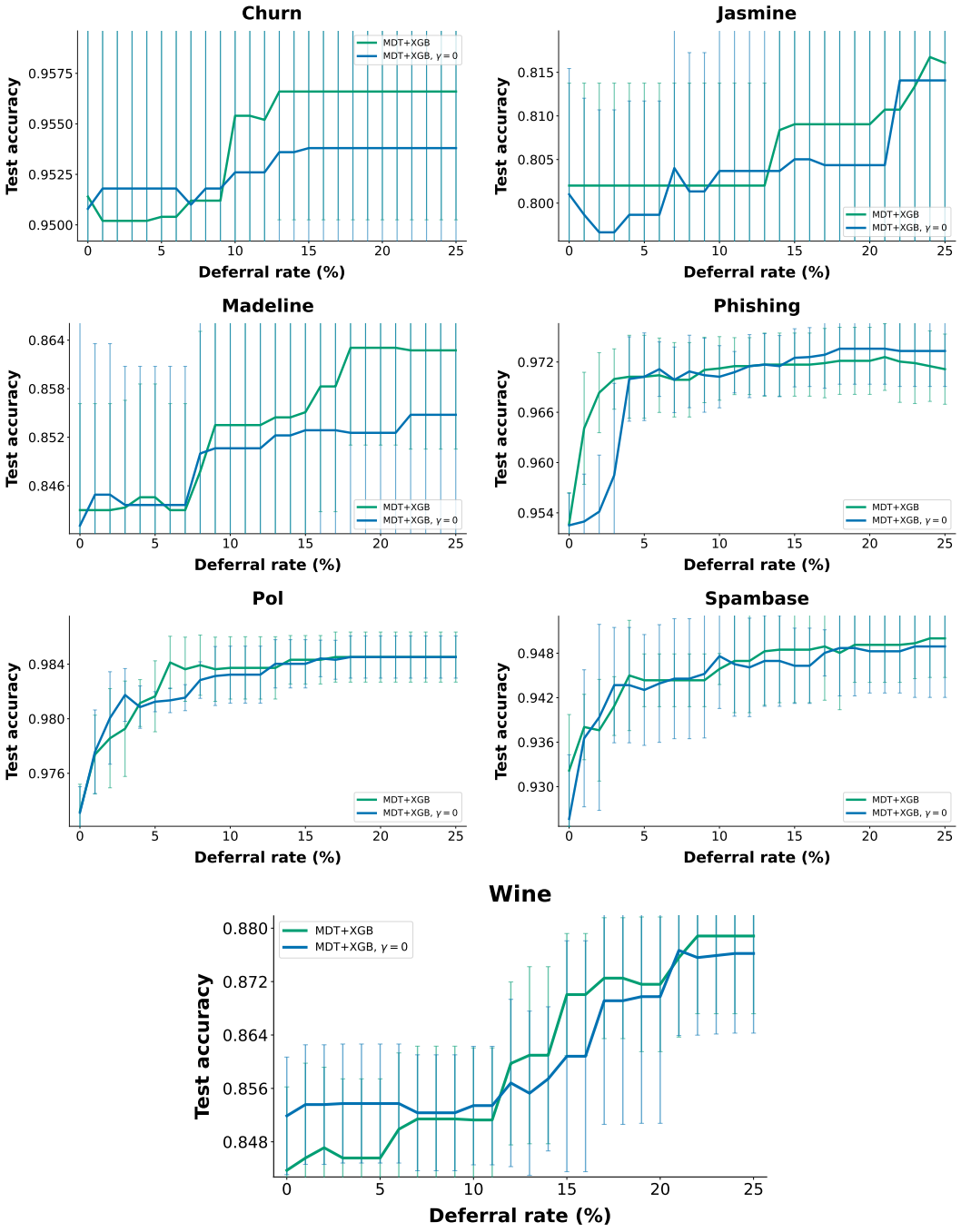


Figure 13: Ablation on the distance-based reweighting via tuning γ or setting $\gamma = 0$.

Dataset	MDT+XGB	MDT+XGB LeaveOneOut
Abalone	0.6333 ± 0.0155	0.6323 ± 0.0128
Adult	0.8681 ± 0.0039	0.8661 ± 0.0026
Aging	0.8028 ± 0.0077	0.8028 ± 0.0077
Bank	0.9090 ± 0.0013	0.9077 ± 0.0021
Bike	0.9493 ± 0.0054	0.9427 ± 0.0029
California	0.9099 ± 0.0049	0.8997 ± 0.0075

Table 11: Ablation of using in-sample black-box predictions versus out-of-fold predictions when training the defer model. Using leave-one-out style out-of-fold predictions does not improve performance and slightly worsens accuracy on most datasets. Results show average test accuracy ($\pm$ standard deviation) under a 50% test deferral constraint during hyperparameter selection.

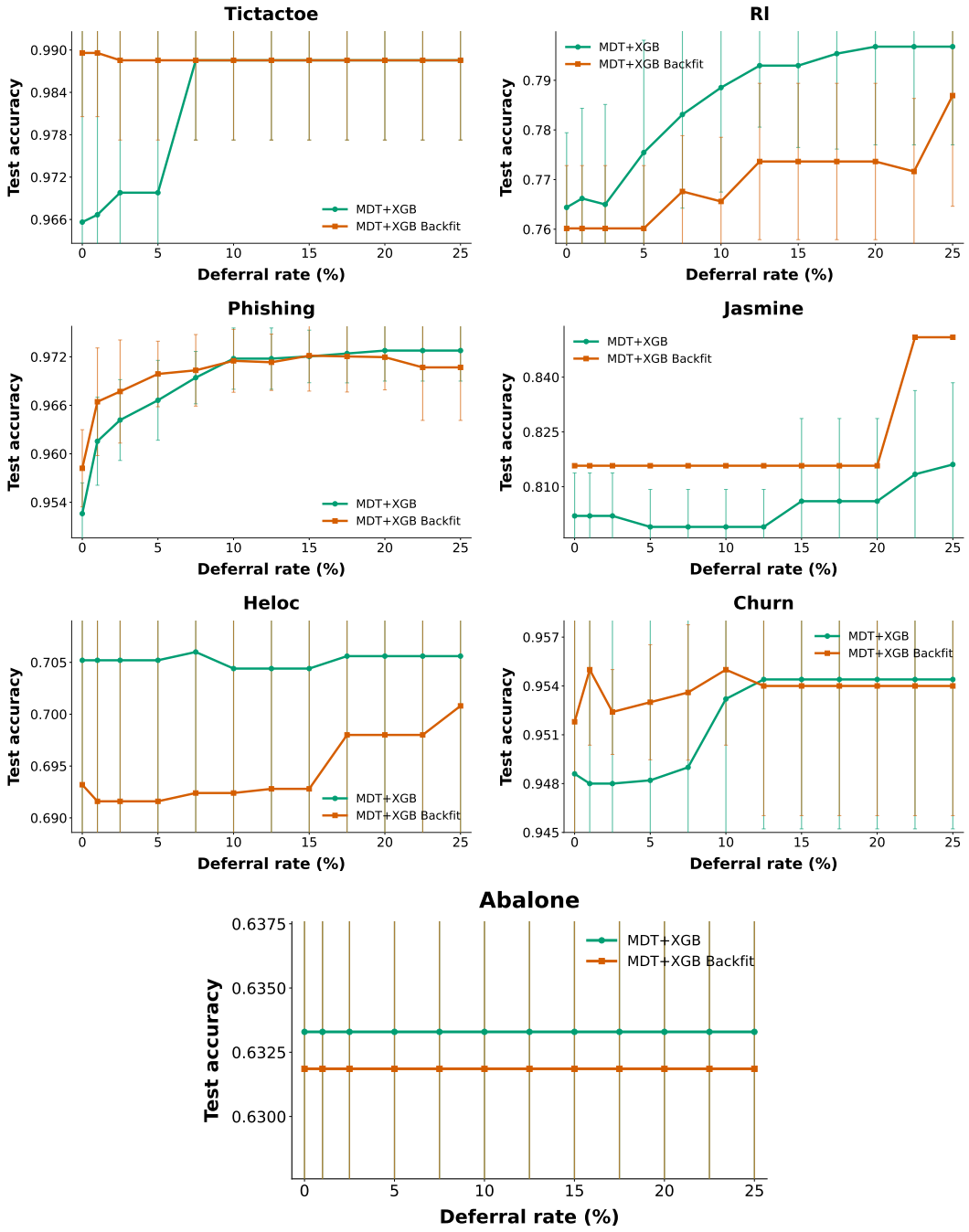


Figure 14: Comparison of MDT+XGB and MDT+XGB with backfitting across selected datasets. Each plot shows test accuracy as a function of the deferral rate. Backfitting gives clear improvements on some datasets, such as Tictactoe, where revisiting stages may help capture higher-order interactions, but it degrades performance on several others.

Table 12: Performance of multistage defer trees (MDTs) trained using a greedy decision tree algorithm operating directly on continuous features. Entries give mean test classification accuracy $\pm$ standard error across five train/test splits (higher is better). $\leq 25\%$ deferral.

Dataset	MDT: greedy (continuous)
Abalone	0.6240 ± 0.0028
Bike	0.9460 ± 0.0024
Churn	0.9544 ± 0.0026
Compas	0.6723 ± 0.0057
Droid	0.9706 ± 0.0005
Monk2	0.9783 ± 0.0086
Phishing	0.9708 ± 0.0018
Pol	0.9824 ± 0.0009
Rl	0.7777 ± 0.0089
Student	0.7923 ± 0.0195
Tictactoe	0.9896 ± 0.0037
Wine	0.8670 ± 0.0023

Table 13: Churn regularization sweep: mean test classification accuracy. Rows vary the sparsity penalty and columns vary the deferral penalty; both penalty values are normalized by the full dataset size N. All other hyperparameters are selected to maximize validation accuracy. Values are means across five train/test splits.

	Deferral penalty								
Sparsity penalty	0.001	0.005	0.01	0.05	0.1	0.3	0.6	0.8	1
0.000010	0.953	0.944	0.935	0.911	0.920	0.919	0.920	0.920	0.920
0.000050	0.956	0.956	0.949	0.939	0.934	0.927	0.925	0.925	0.925
0.000100	0.957	0.957	0.955	0.950	0.944	0.932	0.932	0.932	0.932
0.000500	0.957	0.957	0.957	0.955	0.959	0.948	0.947	0.947	0.947
0.001000	0.957	0.957	0.957	0.956	0.958	0.943	0.943	0.943	0.943
0.005000	0.957	0.957	0.957	0.958	0.950	0.921	0.920	0.920	0.920
0.010000	0.957	0.957	0.957	0.957	0.945	0.902	0.883	0.883	0.883

Table 14: Churn regularization sweep: mean proportion of test examples deferred to the final fallback model. Rows vary the sparsity penalty and columns vary the deferral penalty; both penalty values are normalized by the full dataset size N. All other hyperparameters are selected to maximize validation accuracy. Values are means across five train/test splits.

	Deferral penalty								
Sparsity penalty	0.001	0.005	0.01	0.05	0.1	0.3	0.6	0.8	1
0.000010	0.656	0.221	0.174	0.017	0.010	0.001	0.000	0.000	0.000
0.000050	0.993	0.695	0.273	0.067	0.018	0.001	0.000	0.000	0.000
0.000100	1.000	1.000	0.498	0.064	0.020	0.000	0.000	0.000	0.000
0.000500	1.000	1.000	1.000	0.143	0.090	0.000	0.000	0.000	0.000
0.001000	1.000	1.000	1.000	0.232	0.131	0.001	0.000	0.000	0.000
0.005000	1.000	1.000	1.000	0.875	0.269	0.000	0.000	0.000	0.000
0.010000	1.000	1.000	1.000	1.000	0.299	0.062	0.000	0.000	0.000

Table 15: Churn regularization sweep: mean number of leaves reported for the MDT component. Rows vary the sparsity penalty and columns vary the deferral penalty; both penalty values are normalized by the full dataset size N. All other hyperparameters are selected to maximize validation accuracy. Values are means across five train/test splits.

	Deferral penalty								
Sparsity penalty	0.001	0.005	0.01	0.05	0.1	0.3	0.6	0.8	1
0.000010	73.2	191.0	191.6	261.6	223.4	208.6	207.2	207.2	207.2
0.000050	10.6	30.2	136.6	171.8	179.4	180.4	185.6	185.6	185.6
0.000100	2.0	2.0	69.8	230.0	198.2	136.8	138.4	138.4	138.4
0.000500	2.0	2.0	2.0	86.0	41.4	36.6	34.8	34.8	34.8
0.001000	2.0	2.0	2.0	24.8	17.0	25.8	21.6	21.6	21.6
0.005000	2.0	2.0	2.0	2.8	7.6	14.2	9.0	9.0	9.0
0.010000	2.0	2.0	2.0	2.0	5.0	5.6	3.6	3.6	3.6

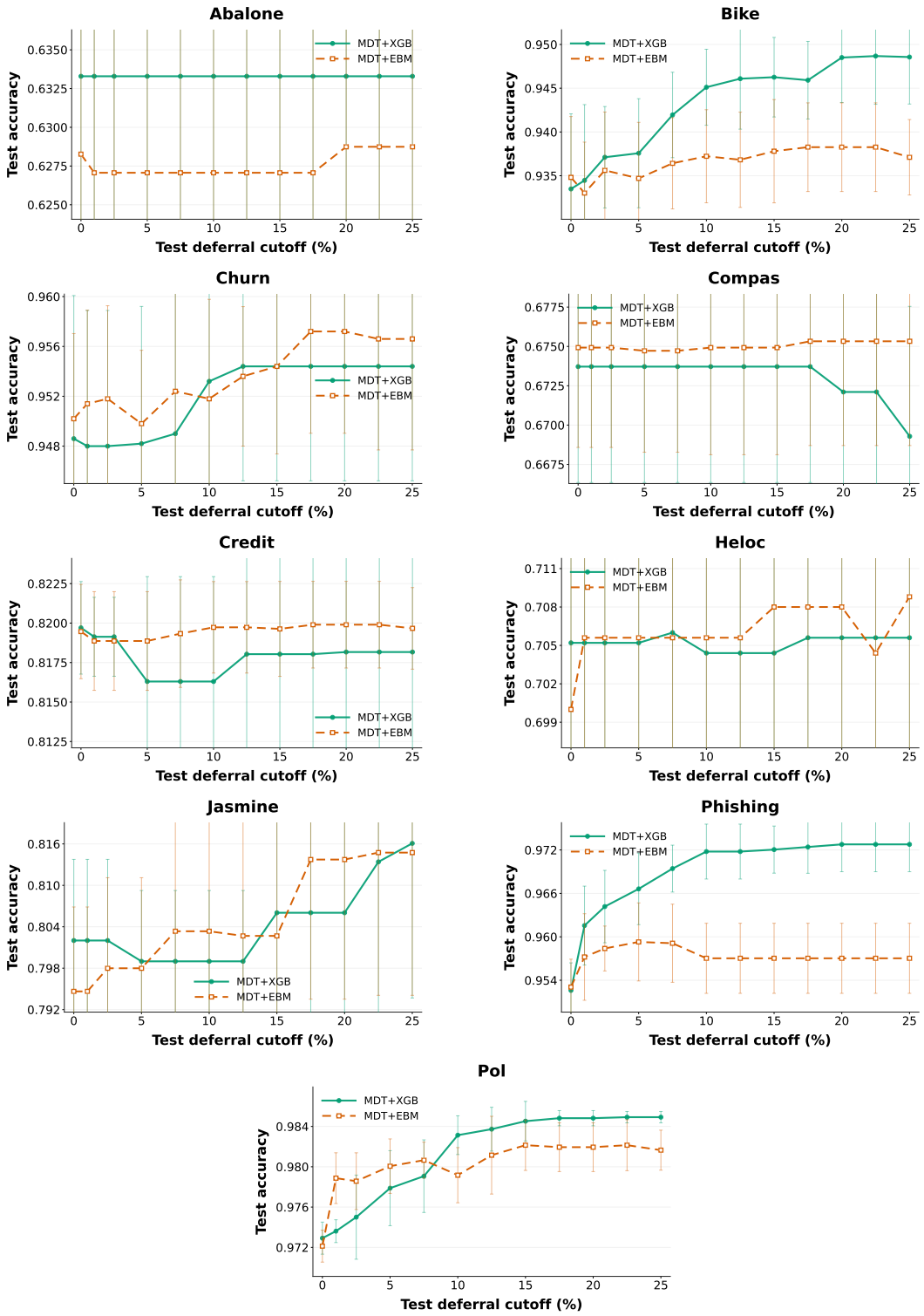


Figure 15: Comparison of MDT+XGB and MDT+EBM across selected datasets. MDT+EBM uses an Explainable Boosting Machine [42] as the fallback model, providing a more interpretable alternative to XGBoost. For some datasets, this amount of fallback complexity is sufficient, and MDT+EBM closely matches MDT+XGB. For other datasets, most notably Bike and Phishing, the EBM fallback is not expressive enough to recover the same accuracy–deferral tradeoff as MDT+XGB.

Table 16: Phishing regularization sweep: mean test classification accuracy. Rows vary the sparsity penalty and columns vary the deferral penalty; both penalty values are normalized by the full dataset size N. All other hyperparameters are selected to maximize validation accuracy. Values are means across five train/test splits.

	Deferral penalty								
Sparsity penalty	0.001	0.005	0.01	0.05	0.1	0.3	0.6	0.8	1
0.000010	0.973	0.971	0.971	0.965	0.961	0.951	0.952	0.952	0.952
0.000050	0.973	0.973	0.974	0.970	0.969	0.954	0.953	0.953	0.953
0.000100	0.974	0.973	0.973	0.972	0.969	0.954	0.952	0.952	0.952
0.000500	0.973	0.974	0.973	0.970	0.965	0.946	0.942	0.942	0.942
0.001000	0.973	0.973	0.974	0.969	0.966	0.943	0.937	0.937	0.937
0.005000	0.973	0.973	0.973	0.970	0.963	0.938	0.914	0.914	0.914
0.010000	0.973	0.973	0.973	0.970	0.943	0.938	0.911	0.911	0.911

Table 17: Phishing regularization sweep: mean proportion of test examples deferred to the final fallback model. Rows vary the sparsity penalty and columns vary the deferral penalty; both penalty values are normalized by the full dataset size N. All other hyperparameters are selected to maximize validation accuracy. Values are means across five train/test splits.

	Deferral penalty								
Sparsity penalty	0.001	0.005	0.01	0.05	0.1	0.3	0.6	0.8	1
0.000010	0.297	0.145	0.110	0.092	0.067	0.005	0.000	0.000	0.000
0.000050	0.598	0.326	0.259	0.088	0.092	0.003	0.000	0.000	0.000
0.000100	0.598	0.440	0.283	0.144	0.083	0.006	0.000	0.000	0.000
0.000500	1.000	0.598	0.577	0.233	0.130	0.005	0.000	0.000	0.000
0.001000	1.000	1.000	0.598	0.262	0.162	0.028	0.000	0.000	0.000
0.005000	1.000	1.000	1.000	0.577	0.310	0.078	0.000	0.000	0.000
0.010000	1.000	1.000	1.000	0.708	0.153	0.078	0.000	0.000	0.000

Table 18: Phishing regularization sweep: mean number of leaves reported for the fitted MDT. Rows vary the sparsity penalty and columns vary the deferral penalty; both penalty values are normalized by the full dataset size N. All other hyperparameters are selected to maximize validation accuracy. Values are means across five train/test splits.

	Deferral penalty								
Sparsity penalty	0.001	0.005	0.01	0.05	0.1	0.3	0.6	0.8	1
0.000010	77.0	203.0	235.2	228.6	285.2	296.4	293.0	293.0	293.0
0.000050	5.4	34.8	59.4	218.2	187.0	230.6	238.2	238.2	238.2
0.000100	5.0	14.8	57.0	133.8	140.8	161.4	139.6	139.6	139.6
0.000500	2.0	5.0	6.6	41.0	89.6	66.4	29.2	29.2	29.2
0.001000	2.0	2.0	5.0	13.6	41.4	37.6	16.4	16.4	16.4
0.005000	2.0	2.0	2.0	4.0	5.6	5.0	4.0	4.0	4.0
0.010000	2.0	2.0	2.0	3.0	4.4	5.0	3.8	3.8	3.8

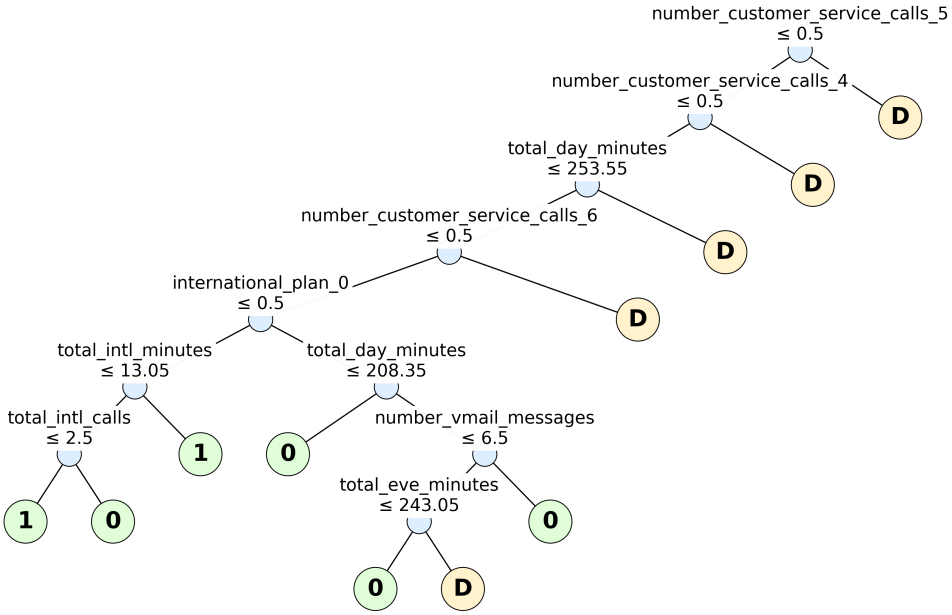
Table 19: Test accuracy of TabPFN [33] and an MDT with TabPFN as the final fallback model, with at most 25% deferral. Entries show mean test accuracy $\pm$ standard error over five train/test splits; higher is better.

Dataset	TabPFN	MDT+TabPFN ($\leq$25%)
Churn	0.972 ± 0.003	0.968 ± 0.004
Compas	0.679 ± 0.004	0.672 ± 0.003
Heloc	0.714 ± 0.010	0.708 ± 0.010
Phishing	0.975 ± 0.002	0.975 ± 0.002
Student	0.815 ± 0.014	0.812 ± 0.012

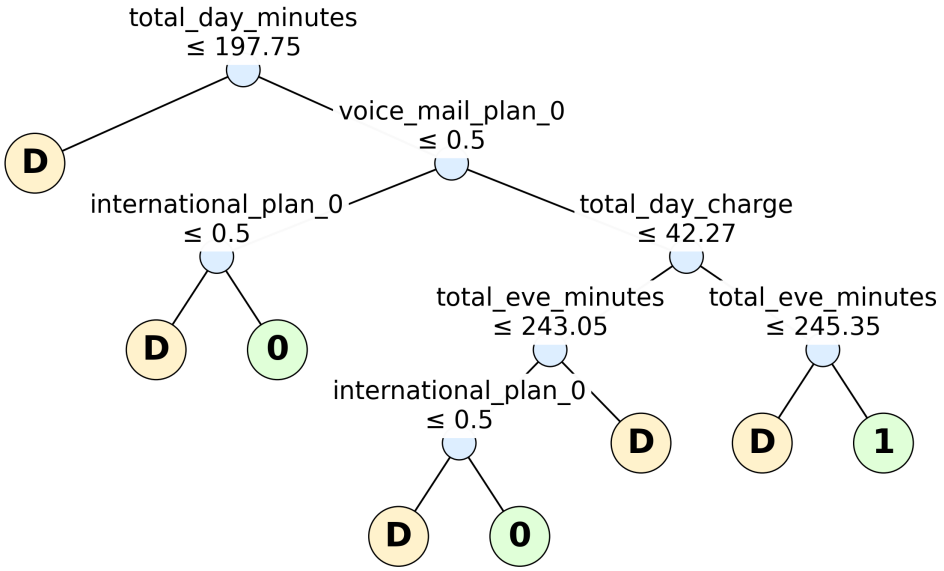
Dataset	Δ Accuracy (pp)	Δ Deferral (pp)
Abalone	$+0.98 \pm 1.65$	-0.19 ± 0.43
Aging	0.00 ± 0.00	0.00 ± 0.00
Bike	$+0.07 \pm 0.17$	$+2.29 \pm 3.94$
Churn	$+0.28 \pm 0.43$	$+1.20 \pm 5.76$
Droid	-0.02 ± 0.12	$+0.25 \pm 3.41$
Heloc	-0.28 ± 0.87	0.00 ± 0.00
Phishing	$+0.02 \pm 0.31$	-0.43 ± 5.08
Pol	$+0.07 \pm 0.22$	-5.34 ± 3.50
RL	-0.08 ± 0.61	$+8.61 \pm 5.89$
Shopping	-0.01 ± 0.13	-5.38 ± 7.62
Spambase	-0.04 ± 0.16	$+0.74 \pm 3.19$
Wine	$+0.02 \pm 0.80$	-2.80 ± 5.27

Table 20: Ablation comparing a fixed binarization derived from XGBoost splits at the root with recomputing the binarization at each stage of the MDT. Entries report the mean $\pm$ standard deviation of differences (fixed binarization minus recomputed) across five train/test splits. For each split, hyperparameters are selected to maximize mean validation accuracy subject to at most 25% test deferral. Positive values indicate higher accuracy or deferral under fixed binarization.

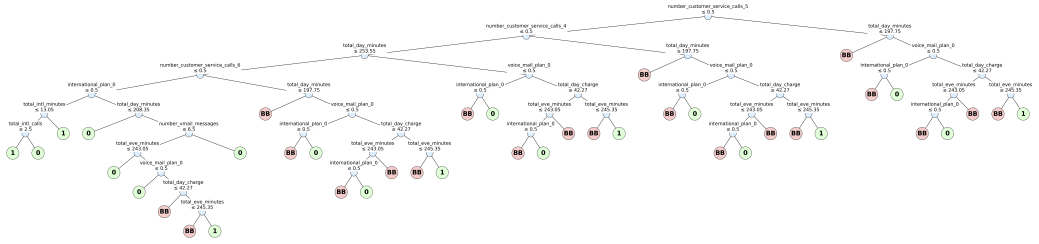
H Examples of MDTs



(a) MDT stage 1

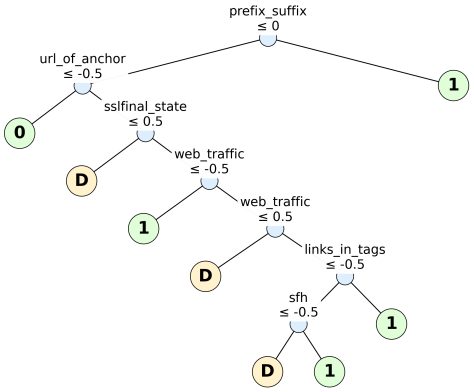


(b) MDT stage 2

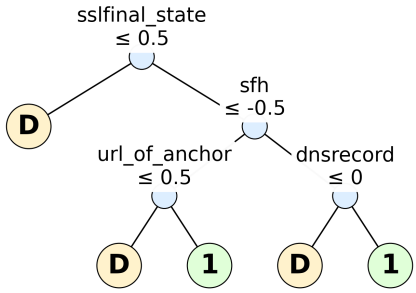


(c) Compressed single-tree representation

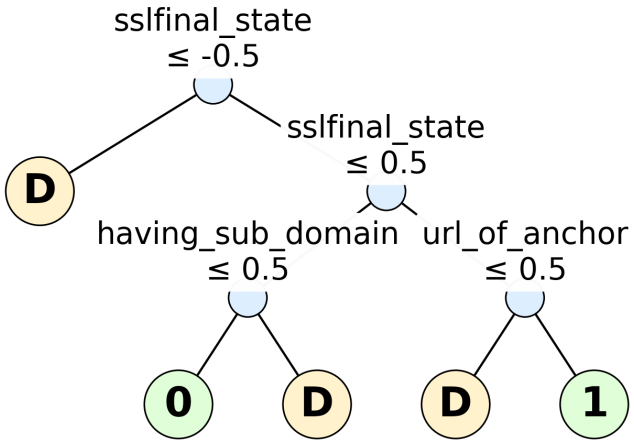
Figure 16: Compressed MDT stages and corresponding compressed single-tree representation for Churn.



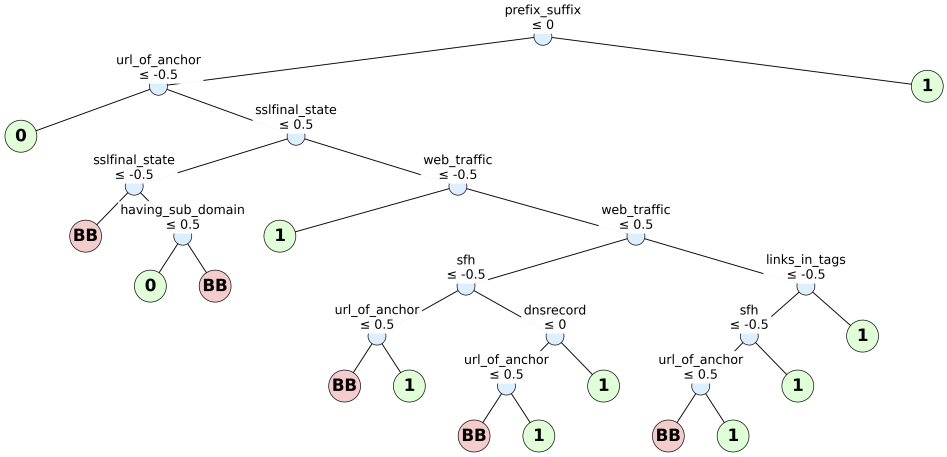
(a) MDT stage 1



(b) MDT stage 2



(c) MDT stage 3



(d) Compressed single-tree representation

Figure 17: Compressed MDT stages and corresponding compressed single-tree representation for Phishing.

Rule Bank

r_1	$\texttt{prefix_suffix} > 0$	$\mapsto 1$
r_2	$\texttt{url_of_anchor} \leq -0.5$	$\mapsto 0$
r_3	$\texttt{sslfinal_state} > 0.5 \wedge \texttt{web_traffic} \leq -0.5$	$\mapsto 1$
r_4	$\texttt{sslfinal_state} > 0.5 \wedge \texttt{web_traffic} > 0.5 \wedge \texttt{links_in_tags} > -0.5$	$\mapsto 1$
r_5	$\texttt{sslfinal_state} > 0.5 \wedge \texttt{web_traffic} > 0.5 \wedge \texttt{sfh} > -0.5$	$\mapsto 1$
r_6	$\texttt{sslfinal_state} > 0.5 \wedge \texttt{sfh} \leq -0.5 \wedge \texttt{url_of_anchor} > 0.5$	$\mapsto 1$
r_7	$\texttt{sslfinal_state} > 0.5 \wedge \texttt{sfh} > -0.5 \wedge \texttt{dnsrecord} > 0$	$\mapsto 1$
r_8	$\texttt{sslfinal_state} > -0.5 \wedge \texttt{sslfinal_state} \leq 0.5 \wedge \texttt{having_sub_domain} \leq 0.5$	$\mapsto 0$
r_9	$\texttt{sslfinal_state} > 0.5 \wedge \texttt{url_of_anchor} > 0.5$	$\mapsto 1$

Sequential Rule List

Stage 1: $r_1 \Rightarrow 1,\ r_2 \Rightarrow 0,\ r_3 \Rightarrow 1,\ r_4 \Rightarrow 1,\ r_5 \Rightarrow 1$, *(move to next stage)*

Stage 2: $r_6 \Rightarrow 1,\ r_7 \Rightarrow 1$, *(move to next stage)*

Stage 3: $r_8 \Rightarrow 0,\ r_9 \Rightarrow 1$, *else defer*

Figure 18: Phishing Rule List Representation.

Broader Impacts Statement

This paper studies interpretable machine learning, a key component of many ethical AI applications. However, the proposed method is general-purpose and is not designed for any specific application.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We state our key contributions in the abstract and introduction. These are: a new model class (MDTs), an iterative approach to training them, and algorithms to write MDTs in other representations.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We generalize a high-quality decision tree algorithm in order to learn a single defer tree. This algorithm does not always yield an optimal defer tree ; additionally, the sequences of sparse decision trees our algorithm constructs are less interpretable than a single sparse decision tree (though we believe the added expressiveness justifies this compromise, it would be ideal to compromise even less).

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Yes, if a theoretical result is stated in the main paper, all of its assumptions are present. If the result is referenced in the main paper but not stated in the main paper, then the formal set of assumptions appears with the corresponding statement in the supplementary material.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide high-level details of hyperparameter selection in the main paper. Appendix F provides a list of datasets, a description of the preprocessing that was done to them, the classification task we perform on them, and the tuning grid for our methods and baselines. Additionally, we will provide a public release of code for our methods when the paper is published.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We will provide open access to the code for our methods if the paper is published. Additionally, all datasets that we used are publicly available and have been cited.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We ran evaluations on 5 train/test splits of each dataset. The full details of hyperparameters and their tuning are in Appendix F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Table 1 and Figure 2 show standard error or standard deviation over 5 train/test splits. Additionally, all 7 tables for experimental results in the appendix show standard deviation across 5 train/test splits.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Table 5 provides timing results for our method on each of the 25 datasets we evaluated on. Appendix F includes information about the machines used. As discussed in Appendix F, we set a time limit of 600 seconds for the two optimal methods we compare to.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have read the NeurIPS Code of Ethics and found no instances where the research does not conform with it.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: Our method is general purpose and not designed for any specific application.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We are not aware of any risks that this paper may pose.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

Justification: We have cited the creators and authors of code and datasets that we used in this work; we have explicitly listed the license for every dataset where we were able to find it (and noted explicitly where we were not able to find a license), and all datasets are available on OpenML.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We provide in-line code comments for our methods. We will provide additional documentation in a public code release after publication.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: We did not perform research with human subjects or perform crowdsourcing experiments.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development does not involve LLMs.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.