Compressing Collections of Trees with Decision Equivalence

Hayden McTavish
Department of Computer Science
Duke University
Durham, NC 27705
hayden.mctavish@duke.edu

Jon Donnelly
Department of Biostatistics
Johns Hopkins University, Duke University
Baltimore, MD 21205
jdonne18@jh.edu

Margo Seltzer
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4
mseltzer@cs.ubc.ca

Cynthia Rudin
Department of Computer Science
Duke University
Durham, NC 27705
cynthia@cs.duke.edu

Abstract

Decision trees with binary leaves are fundamental building blocks in classical machine learning. While individual trees can be used as strong interpretable classifiers, they frequently serve as members of large collections instead: primarily as components of powerful ensembles, but also as one of the few model classes for which the entire set of near-optimal models (the Rashomon set) can be analyzed. Both types of collections are often more complex than necessary, because there are many decision tree representations for the same decision boundary, imposing unnecessary computational costs in ensembles and complicating theoretical work on Rashomon sets. We propose methods to reduce collections of trees to a subset representing each unique decision boundary in the collection, with each tree in its sparsest equivalent form. In so doing, we improve the scalability of removing provably redundant classifiers in a Rashomon set. We also compress many shallow Adaboost ensembles by >15% of their total number of leaves, while exactly preserving the models' decisions and probability estimates on any possible observation. These methods allow practitioners to run more efficient computations over compressed collections of trees and remove arbitrary complexity when analyzing and understanding them.

1 Introduction

Collections of decision tree classifiers are core to classical machine learning. A number of popular tabular classifiers are ensembles of trees [8, 29]. These ensembles consistently rank near the top of tabular benchmarks [26, 30]. They perform well on accuracy as well as metrics of calibration and ranking, such as ROC-AUC [11, 25]. Even when recent work focuses on finding optimal single, interpretable trees as the final classifier instead of an ensemble [1, 12, 18, 36, 45, 54], collections of trees become relevant from a different perspective. Simpler models are most likely to be competitive to black box models when there are many near-optimal models for a dataset [6, 48, 49]. In such settings, it can be of great benefit to materialize and study this set of near-optimal models, called the Rashomon set [8, 46]. One of the few model classes for which the Rashomon set can be materialized and studied exactly is the model class of decision trees [3, 60].

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

It would be ideal to remove redundancies in these collections before using them in downstream tasks. For ensemble classifiers, reducing ensemble size corresponds to improved memory and energy footprint [10]. Compressed ensembles become easier to explain [4] and to verify for properties such as fairness and robustness [20, 34]. For Rashomon sets, a key question relates to disentangling which trees can make distinct predictions on future data and which trees have distinct structure but identical predictions [41, 49]. Frequently, Rashomon sets do contain a large proportion of redundant trees [40], and their removal improves downstream applications such as variable importance [22], but removing these trees scalably remains a challenge.

Collections of decision tree classifiers often exhibit redundancy due to *decision equivalence* [14, 62]; two trees can make the same classification decisions for any possible sample (even one that is out of distribution) despite having distinct structures. This can cause collections to be redundant in two ways. Collections may contain trees exhibiting decision equivalence to each other, which is a frequent occurrence in Rashomon sets [40]. Moreover, individual trees that are trained greedily, such as the trees in many ensembles, can be losslessly reduced to a sparser, decision equivalent tree that corresponds to the exact same function.

We reduce collections of decision trees by removing decision equivalent trees and simplifying each remaining tree to sparsest equivalent form. To remove decision equivalent trees, we present a polynomial-time algorithm that builds a dataset such that every decision equivalence class in the collection has a different classification pattern. To simplify trees to their sparsest form, we create a reduction to a classification task with missing values based on each leaf of the tree and adapt a state-of-the-art optimal decision tree solver framework, STreeD [54], to the task.

This reduced form lets us solve two key problems. First, we can compress classical ensembles, such as Adaboost [29], while exactly preserving the ensemble's ROC-AUC and predicted probabilities. While several existing approaches can compress ensembles while guaranteeing faithful classification decisions [23, 57], they do not guarantee preservation of predicted probabilities or ROC-AUC. Second, we improve the scalability of algorithms to remove decision equivalent forms from a Rashomon set and generalize these checks to identify provable redundancies owing to discretization of continuous features.

Our core contributions are as follows.

1. We provide tools to scalably filter the occurrence of decision equivalence in collections of trees. We demonstrate lossless compression of decision tree Rashomon sets several orders of magnitude faster than prior work (with improved asymptotic complexity) and handle equivalence arising from multiple splits on the same underlying continuous or categorical feature.
2. We provide an algorithm to simplify a decision tree to its sparsest equivalent form, demonstrating practically feasible compression of shallow Adaboost ensembles while exactly preserving their predictions and calibration.

2 Related Work

Decision Equivalence. The fact that many decision tree expressions map to the same decision boundary has a long history of exploration. Cockett [14] outlines an approach to compute equivalence efficiently and provides a complete set of operations to manipulate between decision equivalent forms. Cockett & Herrera [16] then recommend using a subset of these operations to scalably simplify decision trees (though not with guarantees of finding their sparsest equivalent form). Zantema et al. [62] formalize an algorithm for evaluating decision equivalence between a pair of trees with time quadratic in the number of leaves. Zantema et al. [61] go on to prove that finding the sparsest form of a given tree is NP-hard. Decision equivalence has also been explored more recently by various other terms, often in the context of collections of trees. Several such terms are faithful [57], predictively equivalent [40], and functionally identical [23]. Decision equivalence has a number of convenient decompositions in relation to the leaves of trees [14–16]. These leaves correspond to an implicant [43], a weak abductive explanation [32, 33], and a sufficient reason [44] for that tree's prediction. In this work we refer to the subspace that is routed to a particular leaf as a leaf subregion.

Decision Equivalence for Tree Ensembles. In this work, we focus on decision equivalence within ensembles of decision tree classifiers with binary-valued leaves, primarily focusing on Adaboost

[29] (the method also applies to random forests [7]). Both approaches are additive; the predicted probabilities are a (weighted) sum of the predictions of each individual tree. Several works find sparser forms of tree-based ensembles that are guaranteed to make identical classifications for any sample. Vidal & Schiffer [57] present an algorithm to find a single decision tree of minimal size with identical predictions to a random forest of trees, using a branch and bound search over the intersections of leaves in the forest. Emine et al. [23] convert from an additive ensemble to a sparser decision equivalent ensemble by learning a sparse set of weights over the original set of trees. By contrast, our simplification of ensembles preserves both the predicted classes of the ensembles for any possible sample and the predicted probabilities.

Rashomon Sets The Rashomon effect, introduced in machine learning by Breiman [9], refers to the phenomenon whereby there are many nearly equivalently good classifiers for a given machine learning task. The full set of these nearly-equivalent classifiers for a given model class, the Rashomon set, is of interest for a range of applications, including robustness [31], variable importance [21], understanding when simple models will approach the performance of complex models [6, 48, 49], and allowing customization and control to domain experts and practitioners [47]. Decision trees are one of the few model classes for which an exact Rashomon set with respect to regularized empirical risk can be computed relatively tractably [3, 60]. However, some applications, such as variable importance, can be confounded by the existence of decision equivalent trees in the Rashomon set [40]. This confound has motivated researchers to try a range of adaptations, such as clustering trees by predictive behavior on observed data [42] or constraining to a subspace of tree-based partitions where decision equivalence is easier to identify [55].

Two approaches have been proposed to identify decision equivalence in Rashomon sets of decision trees. McTavish et al. [40] discuss ways that accounting for decision equivalence can improve various downstream classification tasks using trees. Several of the paper's investigations relate to compressing Rashomon sets to preserve only one instance of each decision equivalence class. The authors map each tree to a simplified Quine-McCluskey form [43], and then do a linear pass through the Rashomon set to remove trees with identical forms. This scales linearly in the size of the set, but each Quine-McCluskey reduction is NP-hard in the size of the trees, so this is only suitable for the Rashomon sets of shallow trees studied in that work. Marques-Silva & Ignatiev [38] propose using pairwise decision equivalence operations for this task, to allow scaling to larger individual trees. When we are dealing with a Rashomon set, for which the size itself can scale exponentially with the allowed complexity of each component tree, running equivalence checks between every pair in the set can be a substantial computational drawback; but for any collection with complex component trees, a Quine-McCluskey reduction can also be prohibitively expensive. Our work explores scalability improvements for this task that are more efficient per-tree than the approach in [40] and more efficient over the entire set than the approach in [38].

Optimal Decision Tree Classifiers. A number of works globally optimize a decision tree's classification accuracy with a penalty or constraint on the complexity of the corresponding tree. While this has taken many avenues, including mixed integer programming and SAT solvers [5, 56], more recent approaches have found success with branch-and-bound dynamic programming approaches [1, 12, 19, 36, 54]. In this work, we discuss a minor modification to branch-and-bound search strategies for optimal trees to handle missing data, and use this to allow a reduction from finding the sparsest form of a tree to finding an optimal tree on a simple dataset based on the leaves of the original tree.

3 Preliminaries

Define feature space $\mathcal{X} \subseteq \mathbb{R}^p$, where each of the p features is continuous, categorical, or binary. We denote a training dataset with n samples as $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, where each $x_i \in \mathcal{X}$ and each $y_i \in \{0, 1\}$. We focus on binary labels here, but our work extends to multiclass classification with only minor algorithmic adjustments. Define a boolean query $q = (j, t)$ corresponding to the function $\mathbb{1}\{v_j \leq t\}$ (where $\mathbb{1}$ is the indicator function) for an input vector $v \in \mathcal{X}$. Let $queries(T)$ be the list of all unique boolean queries (j, t) in a tree T. A binary decision tree classifies samples by making a series of boolean queries over individual features, routing samples to a left or right subtree based on the query, until reaching a leaf corresponding to a prediction of 0 or 1. Define the class of all binary trees as $\mathcal{T}$. Denote $T(x)$ as the prediction of tree $T \in \mathcal{T}$ for sample x. Let $depth(T)$ be the depth of the tree (the maximum number of queries in any path of the tree), and let the class of trees of depth

at most d be $\mathcal{T}_d = \{T \in \mathcal{T} \mid depth(T) \leq d\}$. Denote the size of the tree T by $|T|$, and let the size correspond to the number of leaves in the tree.

The leaves of decision tree T, denoted as $leaves(T)$, correspond to a partition of the feature space $\mathcal{X}$ based on the queries to reach each leaf. Let $\mathcal{I}(\mathbb{R})$ denote the set of all intervals in $\mathbb{R}$. A leaf subregion $S(\ell) \in \mathcal{I}(\mathbb{R})^p$ corresponds to the subset of the feature space $\mathcal{X}$ that routes to leaf ℓ. For any sample x_i, the sample reaches leaf ℓ if and only if $x_{ij} \in S(\ell)_j$ for every feature $j \in [1, \ldots, p]$. Note that for any features j' not along the root-to-leaf path for leaf ℓ, $S(\ell)_{j'}$ is the open interval $(-\infty, \infty)$. The prediction for any samples reaching leaf $\ell \in leaves(T)$ is denoted $\hat{y}_\ell$.

We define decision equivalence, denoted by $\overset{\text{DE}}{=}$, as a guaranteed correspondence in the predictions of two trees for any possible sample:

Definition 3.1 (Decision Equivalence)**.** $T_1 \overset{DE}{=} T_2 \iff \forall x \in \mathcal{X}, T_1(x) = T_2(x)$,

where we note that decision equivalence is defined independently of the observed data $\mathcal{D}$.

Decision equivalence allows us to define a sparsest form for a tree (with a constraint on depth to keep our optimization task feasible and to prevent increases to the worst case inference time of a tree).

Definition 3.2 (Sparsest Form)**.** *Let T be some decision tree. For some depth d, denote the set of all sparsest forms of tree T in $\mathcal{T}_d$ as*

$$\textsc{Sp}(T, d) := \underset{\{T' \in \mathcal{T}_d | T' \overset{DE}{=} T\}}{\arg\min} |T'|$$

Given some collection $C_d \subseteq \mathcal{T}_d$, we can define its reduced form as having only one tree, in sparsest form, for each decision equivalence class in C_d:

Definition 3.3 (Reduced Form)**.**

$$\textsc{Reduced}(C_d) := \underset{C'_d \subseteq \mathcal{T}_d}{\arg\min} \{|C'_d| \textit{ s.t. } \forall T \in C_d, \exists T' \in C'_d, T' \in \textsc{Sp}(T, d)\}.$$

Completions. To represent how queries are evaluated on a leaf subregion, it is convenient to work in a feature space of booleans that may be unknown.

Let $M \subseteq \{0, 1, NA\}^q$ be the space of boolean features with potentially unknown (NA) values. A dataset in this space of size n' is then $\mathcal{D}_{missing} = \{(m_{i'}, y_{i'})\}_{i'=1}^{n'}$ with feature vectors $m_{i'} \in M$. A Completion of $m_{i'}$ is a possible replacement of each NA value in the feature vector with 0 or 1. Define the set of all such completions as:

$$\textsc{Completions}(m_{i'}) := \{z_{i'} \in \{0, 1\}^q \mid \forall j \in [1, \ldots, q], z_{i'j} = m_{i'j} \vee m_{i'j} = NA\}.$$

Similarly, define a completion of a whole dataset as any possible completion of each m_i. So the set of all possible completions of a dataset $D_{missing}$ is:

$$\textsc{Completions}(\mathcal{D}_{missing}) := \{\{(z_{i'}, y_{i'})\}_{i'=1}^{n'} \mid \forall i', z_{i'} \in \textsc{Completions}(m_{i'})\}.$$

Rashomon Sets. A Rashomon set for some observed dataset $\mathcal{D}$ describes the set of all models whose objective falls within some specified tolerance ε of the optimum. For the model class of decision trees, Rashomon sets are often defined with respect to the following objective [3, 60]:

$$\text{Obj}(T, \mathcal{D}, \lambda) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{T(x_i) \neq y_i\} + \lambda |T|, \tag{1}$$

for some leaf penalty λ. The Rashomon set for objective tolerance ε and depth constraint d is then:

$$\text{Rset}_\varepsilon(\mathcal{D}, \lambda, d) = \{T \in \mathcal{T}_d \mid \text{Obj}(T, \mathcal{D}, \lambda) \leq \min_{T' \in T_d} \text{Obj}(T', \mathcal{D}, \lambda) + \varepsilon\}. \tag{2}$$

4 Methods

We now outline our methods that find reduced forms for collections of trees (Definition 3.3). This decomposes into two tasks: preserving only one tree from each set of decision equivalent trees (Section 4.1) and reducing each tree to its sparsest equivalent form (Section 4.2).

4.1 Identifying Decision Equivalence in a Collection

Given a collection of trees C_d, we would like to preserve only one tree per decision equivalence class:

$$\arg\min_{C'_d \subseteq \mathcal{T}_d} \{|C'_d| \text{ s.t. } \forall T \in C_d, \exists T' \in C'_d, T' \overset{\text{DE}}{=} T\} \tag{3}$$

To accomplish this, we iterate through a collection of trees, adding trees one at a time to a new collection, and skipping trees for which a decision equivalent tree already exists. To minimize the total number of decision equivalence comparisons we need to make, we maintain a dataset of samples for which no two trees in the current collection have exactly the same prediction vector. This ensures that each new tree we add requires at most one decision equivalence check.

Algorithm 1 presents the approach. After initializing a hashmap PRED_MAP to store the trees we will keep in our compressed collection, and a list of samples $X^{(\text{diff})}$, we iterate through the collection of trees (C_d), maintaining the invariant that we can use the predictions on $X^{(\text{diff})}$ to separate all the decision equivalence classes we have seen so far. For each tree in C_d, we check if its predictions on $X^{(\text{diff})}$ correspond to a tree already in PRED_MAP (lines 6-9). If we do have a collision for all samples in $X^{(\text{diff})}$, we check if the two trees are decision equivalent (line 11). If there is a data point for which the trees differ, then we update the colliding tree and the new tree's key, add both to C' (lines 12-16), and update $X^{(\text{diff})}$. If the trees were decision equivalent, then we can safely leave T out of the compressed set and only keep T' (line 16). If a tree T never hits a collision in PRED_MAP, then it is guaranteed not to be equivalent to any tree currently in PRED_MAP, so we can add it without modifying $X^{(\text{diff})}$ (line 19).

Algorithm 1 SIMPLIFYCOLLECTION(C_d)

Input: Collection of trees C_d.
Output: A compressed version of C_d satisfying Equation (3); only one member of each decision equivalence class is preserved.

1: $X^{(\text{diff})} \leftarrow []$ {dataset to distinguish between trees in our collection}
2: PRED_MAP $\leftarrow \{\}$ {Map predictions on $X^{(\text{diff})}$ to a tree in the compressed set}
(Invariant: a decision equivalence class's index into PRED_MAP is always a prefix of predictions on $X^{(\text{diff})}$ that is unique from all other equivalence classes in PRED_MAP.)
3: **for** $T \in C_d$ **do**
{Start by finding prefix of predictions on $X^{(\text{diff})}$ in PRED_MAP, if one exists: }
4: **if** $\exists k \leq |X^{(\text{diff})}|$ s.t. $[T(x_i)$ for $x_i \in X^{(\text{diff})}_{1:k}] \in$ PRED_MAP **then**
5: potential_collision $\leftarrow [T(x_i)$ for $x_i \in X^{(\text{diff})}_{1:k}]$
6: T' $\leftarrow$ PRED_MAP.*pop*(potential_collision) {remove collision from PRED_MAP}
7: **if** $\exists k'$ s.t. $|X^{(\text{diff})}| \geq k' > k, T(x^{(\text{diff})}_{k'}) \neq T'(x^{(\text{diff})}_{k'})$ **then**
{If there are multiple k', any work; our approach uses the smallest}
8: PRED_MAP$[T(x_i)$ for $x_i \in X^{(\text{diff})}_{1:k'}] = T$
9: PRED_MAP$[T'(x_i)$ for $x_i \in X^{(\text{diff})}_{1:k'}] = T'$
10: **else**
11: x $\leftarrow$ FINDNONEQUIVALENCE(T, T') {Algorithm 5; returns an x s.t. $T(x) \neq T'(x)$}
12: **if** x $\neq$ None **then**
13: $X^{(\text{diff})} \leftarrow X^{(\text{diff})}$+[x] {Augment $X^{(\text{diff})}$ to maintain invariant}
14: PRED_MAP$[T(x_i)$ for $x_i \in X^{(\text{diff})}] = T$ {collision resolved and $T \overset{\text{DE}}{\neq} T'$, so add T}
15: **end if**
16: PRED_MAP$[[T'(x_i)$ for $x_i \in X^{(\text{diff})}]] = T'$ {add T' back in since we popped it earlier}
17: **end if**
18: **else**
19: PRED_MAP$[T(x_i)$ for $x_i \in X^{(\text{diff})}] \leftarrow T$ {no collision in PRED_MAP}
20: **end if**
21: **end for**
22: Return PRED_MAP.values()

Algorithm 5 in the appendix presents the pairwise equivalence subroutine we use. If the two provided trees are not equivalent, the algorithm returns a data point for which the trees' predictions differ, which helps in constructing $X^{(\text{diff})}$. If the two trees are equivalent, the algorithm returns None. We defer this algorithm to the appendix, because there exist algorithms with similar scalability [14, 62]. The key difference in our algorithm is that we can handle equivalences where boolean splits relate to the same underlying continuous feature, at no additional asymptotic cost. Section 5 demonstrates that, for several Rashomon sets trained on real-world data, these continuous feature dependencies account for a substantial proportion of all decision equivalence.

Proposition 4.1, proven in Appendix B, shows that the algorithm is correct and provides an upper bound on its runtime. If the collection size is sufficiently large, then a simple upper bound for the runtime becomes $\mathcal{O}(d|C_d|^2)$. The approach gives a substantial improvement on the asymptotics of existing approaches. It removes the need for the NP-hard subroutine used by McTavish et al. [40]. It also avoids the $\mathcal{O}(|C_d|^2)$ pairwise checks needed for a naïve compression, where each pairwise check would require time quadratic in the leaves of the two compared trees.

Proposition 4.1. *Let the original collection be denoted C_d, and let the domain of the trees have dimensionality p. Let the output of Algorithm 1 applied to C_d be C'_d. Then, C'_d satisfies equation (3), and the algorithm has runtime in*

$$\mathcal{O}\left(|C_d|\left(|C'_d|d + \max_{T\in C_d, T'\in C'_d} |T||T'| + p\right)\right).$$

Reducing Rashomon sets. A recent algorithmic innovation allows decision tree Rashomon sets to be enumerated in sorted order of objective [3]. If we traverse a collection in this sorted order and preserve only the first tree encountered for each decision equivalence class, then Remark 4.2 guarantees the result is in Reduced Form (Definition 3.3), without the need to run a separate sparsest form solver.

Remark 4.2. *Let $R = \mathit{Rset}_\epsilon(\mathcal{D}, \lambda, d)$ be a Rashomon set for some dataset $\mathcal{D}$, slack ϵ, maximum depth d, and leaf penalty $\lambda > 0$. Then its reduced form corresponds to preserving only one tree of lowest objective within each decision equivalence class.*

$$\arg\min_{C'_d \subseteq R}\{|C'_d| \text{ s.t. } \forall T \in R, \exists T' \in C'_d, T' \overset{DE}{=} T, \mathit{Obj}(T', \mathcal{D}, \lambda) \leq \mathit{Obj}(T, \mathcal{D}, \lambda)\} \in \textsc{Reduced}(R).$$

4.2 Reducing Trees to Sparsest Form

Given a decision tree T_{original}, we would like to find a decision equivalent tree of minimal size (that is, a member of $\text{SP}(T_{\text{original}}, d)$ from Definition 3.2). While this problem is NP-hard [62], we can leverage recent advances in optimal sparse decision tree solvers to find sparsest forms for small trees tractably. Ordinarily, an optimal sparse decision tree solver [36, 54] would find a minimizer for objective (1) from Section 3. Finding the sparsest form of a tree is a distinct task: we want a tree that predicts exactly the same as the original tree on a potentially infinite set of points (any $x \in \mathcal{X}$), while being as sparse as possible. To accomplish this, we introduce two modifications: we show how to create a special dataset that partitions the space $\mathcal{X}$ according to the leaves of T_{original}, and we make a small adjustment to the partitioning behavior of an optimal solver to handle this dataset appropriately.

Leaf dataset. In Algorithm 2, we outline a way to construct a compact dataset of size $|T_{\text{original}}| \times |\text{queries}(T_{\text{original}})|$, such that each training sample corresponds to a leaf in tree T_{original}. Each column corresponds to one of the queries in tree T_{original}, and a row of the dataset evaluates to 1, 0, or NA for each query depending on whether the query holds for all possible samples routed to the leaf corresponding to that row. A tree that correctly classifies all possible completions of this dataset corresponds to a decision equivalent tree (see Lemma 4.3, proven in the appendix).

Lemma 4.3. *Let $\mathcal{D}_{missing} = \{(m_{i'}, y_{i'})\}_{i'=1}^{|T'|}$ be the output of Algorithm 2 for tree T'. If the worst-case classification error of tree T across any completion is 0, then $T \overset{DE}{=} T'$. If no path in T or T' is impossible (contains a query with value already known from earlier queries along that path), and queries are all of the form $\mathbb{1}\{x_{ij} \leq t\}$, then this becomes bidirectional:*

$$T \overset{DE}{=} T' \iff \left(\sum_{i'=1}^{|T'|} \max_{x_{i'} \in Completions(m_{i'})} \mathbb{1}\{T(x_{i'}) \neq y_{i'}\}\right) = 0$$

Algorithm 2 LEAFDATASET(T)

Input: Decision tree T
Output: Dataset $X_T \in \{0, 1, \text{NA}\}^{|T| \times |queries(T)|}$, $y_T \in \{0, 1\}^{|T|}$ for which perfect classification on all possible completions of this dataset is necessary and sufficient for a tree to be decision equivalent to T.

```
X_T ← []
y_T ← []
for ℓ ∈ leaves(T) do
    x_T ← []
    for k ∈ |queries(T)| do
        j, t ← queries(T)[k] {Collect j, t from query of form 𝟙[x_ij ≤ t]}
        if S(ℓ)_j ⊆ (−∞, t) then
            x_T.append(1) {we know ∀x_i ∈ S, x_ij ≤ t, so append 1}
        else if S(ℓ)_j ⊆ (t, ∞) then
            x_T.append(0) {we know ∀x_i ∈ S, x_ij > t, so append 0}
        else
            x_T.append(NA) {either answer for x_ij ≤ t is possible}
        end if
    end for
    X_T.append(x_T)
    y_T.append(ŷ_ℓ)
end for
Return X_T, y_T
```

Modified STreeD. Algorithm 3 shows the key adjustment made to an optimal tree solver: when a query cannot be evaluated for some sample (i.e., the value for that query is NA), we send the sample down both paths of the tree instead of one, penalizing either subtree for misclassifying that sample. Adding this modification to STreeD produces an algorithm that finds a tree that minimizes an upper bound on errors for the worst-case completion of a dataset with missing values (Lemma 4.4). In particular, this means that if the modified algorithm reports a tree with 0 misclassification error for some dataset, that tree is guaranteed to have 0 error on any completion of the dataset.

Lemma 4.4. *Let $T^{mod}_{\mathcal{D},d,\lambda}$ be a tree output from an optimal decision tree solver modified with Algorithm 3. Instead of being optimal for objective (1), this tree is optimal for an upper bound on Equation (1) for the worst-case completion of a dataset, subject to a sparsity penalty:*

$$T^{mod}_{\mathcal{D},d,\lambda} \in \operatorname*{arg\,min}_{T \in \mathcal{T}_d} \left(\lambda |T| + \sum_{\ell \in leaves(T)} \max_{\{(x_{i'}, y_{i'})\}_{i'=1}^{|\mathcal{D}|} \in Completions(\mathcal{D})} \sum_{i'=1}^{|\mathcal{D}|} \mathbb{1}\{x_{i'} \in S(\ell) \wedge y_{i'} \neq \hat{y}_\ell\} \right)$$

Taken together, Lemmas 4.3 and 4.4 tell us that we can compose the modified solver and the modified dataset to find a sparsest form (Theorem 4.5). Finding sparsest forms allows us to faithfully reduce ensembles that use binary leaf predictions in their probability estimates and predictions (Remark 4.6).

Theorem 4.5. *Algorithm 4 returns a sparsest decision equivalent form of the input tree within the specified depth bound, if one exists.*

Remark 4.6. *Consider any ensemble of m binary leaf decision trees $C = [T^{(1)}, \dots T^{(m)}] \in \mathcal{T}_d^m$ and weights $[w^{(1)}, \dots, w^{(m)}]$. If the ensemble's probability prediction is*

$$\hat{p}(x) = \frac{1}{\sum_{i=1}^m w^{(i)}} \sum_{i=1}^m w^{(i)} T^{(i)}(x),$$

Then its predicted probabilities are exactly equivalent for the alternative ensemble $[T'^{(1)}, \dots T'^{(k)}] =$ REDUCED(C), with weights $[w'^{(1)}, \dots, w'^{(k)}]$ such that $w'^{(j)} = \sum_{i=1}^m w^{(i)} \mathbb{1}\{T^{(i)} \overset{DE}{=} T'^{(j)}\}$.

Algorithm 3 CONSERVATIVEPARTITION($\mathcal{D}, j$)

Require: Dataset $\mathcal{D}$ where each feature has values $\in \{0, 1, \text{NA}\}$, feature j
Ensure: Resulting partitions $\mathcal{D}_{\text{left}}, \mathcal{D}_{\text{right}}$, for which to evaluate accuracy and optimize subtrees.

$\mathcal{D}_{\text{left}} \leftarrow \{(x_i, y_i) \in \mathcal{D} \mid x_{ij} = 1\} \cup \{(x_i, y_i) \in \mathcal{D} \mid x_{ij} = \text{NA}\}$
$\mathcal{D}_{\text{right}} \leftarrow \{(x_i, y_i) \in \mathcal{D} \mid x_{ij} = 0\} \cup \{(x_i, y_i) \in \mathcal{D} \mid x_{ij} = \text{NA}\}$
Return $\mathcal{D}_{\text{left}}, \mathcal{D}_{\text{right}}$

Algorithm 4 SparsestFormSolver(T, d)

Require: Tree T, search depth d

STREED$_{\text{conservative-partition}} \leftarrow$ STREED modified to use CONSERVATIVEPARTITION (Algorithm 3) in place of the normal partition algorithm
$\mathcal{D}_{sparse} \leftarrow$ LEAFDATASET(T) {Algorithm 2}
solver $\leftarrow$ STREED$_{\text{conservative-partition}}$(depth $= d$, regularization $= \frac{1}{|T|^2+1}$)
solver.fit($\mathcal{D}_{sparse}$)
Return solver.tree

5 Experiments

5.1 Rashomon set Compression

Figure 1 demonstrates the scalability of our Rashomon compression (Section 4.1) relative to the baselines suggested by [38, 40]. We see that our approach scales substantially better in both the complexity of trees and the size of the set. We can handle complex individual trees much better than [40], and we can handle larger Rashomon sets more scalably than [38]. We also demonstrate in Table 1 that accounting for continuous feature dependencies further shrinks the Rashomon set. Revisiting datasets and configurations from [40] that use continuous features, some (though not all) datasets obtain 2-3x more compression of provably equivalent trees, undetectable to previous methods. These results demonstrate that decision equivalence in the Rashomon set is even more pervasive than previously thought and provide a method to scalably **compress the Rashomon set by an order of magnitude**.

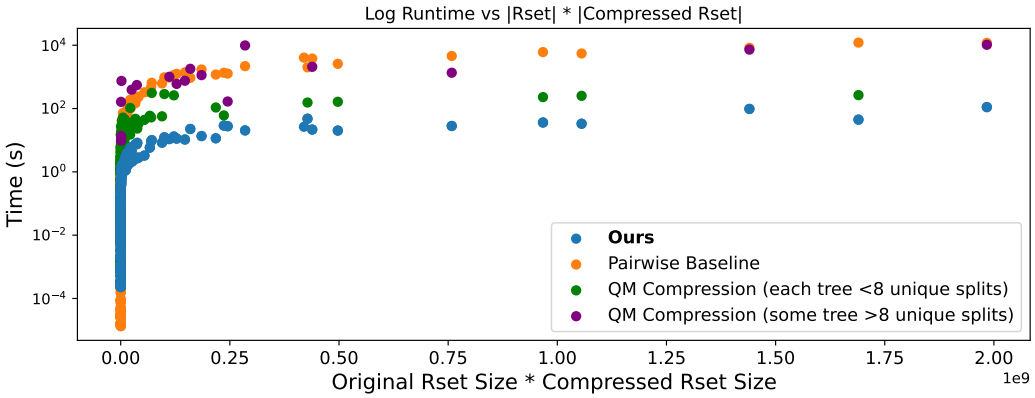


Figure 1: Time to simplify the Rashomon set, using Algorithm 1 (Ours), with pairwise equivalence checks (Pairwise Baseline), or with logically simplifying each tree (QM Compression [40]). Results aggregated for all datasets in our suite, and all Rashomon sets of depth bound $[4, 5]$, epsilon $[0.02, 0.05]$, regularization $[0.01, 0.005]$ that terminate within 24 hours and with 100GB of memory allowed. Our approach is several orders of magnitude faster, particularly for large, diverse Rashomon sets with trees that reason on more than a few splits.

Dataset	Total Trees	w/o Trivial [60]	Boolean DE [40]	Ours
COMPAS	12785 ± 2840	3913 ± 837	2135 ± 448	641 ± 128
Wine Quality	6936 ± 700	2341 ± 377	1409 ± 256	436 ± 90
Wisconsin	24052 ± 9278	11990 ± 5283	4657 ± 1893	4267 ± 1728
FICO	8086 ± 2202	1969 ± 643	1154 ± 375	752 ± 254

Table 1: Total # of trees in the Rashomon set, and # of trees with methods to address equivalence: removing trivial redundancies (a filter provided in the original tree Rashomon set paper [60]), removing decision equivalent trees assuming all boolean feature assignments are possible [40], and removing decision equivalent trees accounting for continuous features (ours). Averaged over 5 folds.

5.2 Adaboost Ensemble Compression

As a case study for sparsest form compression, we explore reducing the representation of Adaboost [29] classifiers. Concretely, this means identifying all trees in the same decision equivalence class and replacing them with a single tree that is in sparsest form, and has weight equal to the sum of weights of all the trees in its equivalence class. As discussed in Remark 4.6, this exactly preserves calibration of the original model. We demonstrate this reduction results in substantial compression of the classifier and that the algorithm is practically feasible for shallow ensembles.

Compression. Table 2 shows the compression achieved between the reduced form and the original model. Because each tree is trained greedily, it is often quite far from the sparsest decision equivalent form. Occasionally, there are also multiple decision equivalent classifiers in the ensemble (these do not always occur but are sometimes significant), which can have their weight shifted to just one member of the equivalence class without changing the classifier's performance.

Dataset	Compression ratio	Proportion of Unique Trees
Iris_setosa	1.000 ± 0.000	1.000 ± 0.000
Iris_virginica	0.740 ± 0.155	0.810 ± 0.163
Iris_versicolor	0.727 ± 0.109	0.828 ± 0.127
Spiral	0.848 ± 0.009	0.958 ± 0.010
Tic-Tac-Toe	0.821 ± 0.005	1.000 ± 0.000
Broward	0.743 ± 0.024	0.852 ± 0.033
COMPAS	0.196 ± 0.048	0.246 ± 0.057
Wisconsin	0.972 ± 0.002	1.000 ± 0.000
FICO	0.776 ± 0.030	0.878 ± 0.031
Wine Quality	0.790 ± 0.016	0.942 ± 0.011
Coupon	0.764 ± 0.005	0.998 ± 0.002
Netherlands	0.441 ± 0.060	0.504 ± 0.085

Table 2: Compression ratio ($\frac{\text{compressed ensemble leaf count}}{\text{original ensemble leaf count}}$) of an ensemble of 100 depth 3 estimators learned by Adaboost. Error bars are standard error from 5 folds. The full compression (both Sections 4.1 and 4.2) is above 15 % for most datasets. Even removing only equivalent pairwise forms without using the sparsest form finder results in some reduction in the number of trees.

There are two cases where compression is minor relative to the other datasets. Iris_setosa, the only dataset without compression, is a very easy one-vs-rest problem where perfect accuracy is achieved by Adaboost with an early exit on a single decision stump. Note that nontrivial decision stumps are already in sparsest form. For Wisconsin, the other dataset with low compression (no duplicate trees and a small compression ratio), many relevant feature dependencies exist that are out of scope for this work. Important features in this dataset, like the area, radius, and perimeter of a cell nucleus, clearly have dependencies that would warrant some dataset-specific modeling. This paper will miss out on compression where trees have distinct behaviour only in impossible or nearly impossible regions that depend on multiple different features, such as when area is high but perimeter is low. While we chose to focus in this work on only broader dependencies that can be modeled generally, exactly, and scalably across datasets (such as age <30 implying age <40), there is a clear direction for future work to model these additional dependencies approximately or exactly.

Runtime. Table 3 shows that our method efficiently compresses the trees in our ensembles. Most significantly, **we can frequently reduce Adaboost's complexity by >15% in under a second**, reducing it to the reduced form defined in Definition 3.3.

In the appendix, we show that using 100 depth 3 classifiers is comparable to the best tuned selection of hyperparameters for these datasets.

Dataset	REDUCED (Sections 4.1 & 4.2)	Remove DE trees (Section 4.1)
Iris_setosa	0.001 ± 0.000	0.000 ± 0.000
Iris_virginica	0.161 ± 0.066	0.012 ± 0.005
Iris_versicolor	0.254 ± 0.064	0.038 ± 0.024
Spiral	0.317 ± 0.023	0.018 ± 0.000
Tic-Tac-Toe	0.529 ± 0.020	0.023 ± 0.000
Broward	0.323 ± 0.027	0.025 ± 0.001
COMPAS	0.066 ± 0.010	0.018 ± 0.001
Wisconsin	0.789 ± 0.033	0.025 ± 0.001
FICO	0.422 ± 0.055	0.024 ± 0.001
Wine Quality	0.483 ± 0.025	0.048 ± 0.026
Coupon	0.557 ± 0.031	0.032 ± 0.000
Netherlands	0.215 ± 0.038	0.021 ± 0.002

Table 3: Runtime in seconds for compressing an ensemble of 100 depth 3 estimators learned by Adaboost. Error bars are standard error from 5 folds. The whole compression takes under a second.

6 Conclusion

We have presented algorithms for losslessly compressing a collection of decision trees for binary classification. When there are multiple trees in the collection that make equivalent classifications for any possible input, we preserve only a single, sparsest tree from this equivalence class. We demonstrate this compression can be done scalably and completely for Rashomon sets and that we can compress ensembles while preserving their exact calibration for any possible sample. Future work could extend this compression to cases beyond binary classification trees, such as ensembles of trees with continuous leaf values [13] (one approximate way is to relax the decision equivalence check in Algorithm 5 to instead verify that two trees' predictions cannot deviate by more than some small amount). Further efficiency improvements in reducing trees to their sparsest forms could allow this approach to scale to deeper ensembles. It may also be of interest to combine this work with other compression methods: Algorithm 1 can be run as a quick preprocessing step to simplify a collection exactly before running further compression methods; and Algorithm 4 can be run on each remaining tree in the output of a compressed ensemble to convert each tree to its sparsest equivalent form. The code used for experiments is available at `https://github.com/HaydenMcT/decision-equivalent-tree-compression`.

7 Acknowledgments

This work relates to Department of Navy award N000142612253 issued by the Office of Naval Research. We acknowledge funding from the National Institutes of Health (NIH) under grant R01-DA054994. Additionally, this material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE 2139754. 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 would also like to thank Elif Arslan for comments helpful to implementing the STreeD package modification to support conservative partitioning (Algorithm 3), as well as Zachery Boner and Zakk Heile for comments on earlier iterations of the work.

References

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

pp. 3146–3153, 2020.

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

[3] Arslan, E., van der Linden, J. G. M., Hoogendoorn, S., Rinaldi, M., and Demirović, E. Sorted rashomon sets of sparse decision trees: Anytime enumeration. *NIPS*, 2025. URL `https://www.paperdigest.org/?paper_id=nips-forum-id-Gibq7Wa7Bq-2025-11-07`.

[4] Bassan, S., Amir, G., Zehavi, M., and Katz, G. What makes an ensemble (un) interpretable? *arXiv preprint arXiv:2506.08216*, 2025.

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

[6] Boner, Z., Chen, H., Semenova, L., Parr, R., and Rudin, C. Using noise to infer aspects of simplicity without learning. In *Advances In Neural Information Processing Systems*, 2024.

[7] Breiman, L. *Classification and Regression Trees*. Routledge, 1984.

[8] Breiman, L. Random forests. *Machine Learning*, 45:5–32, 2001.

[9] Breiman, L. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199–231, 2001.

[10] Buschjäger, S. and Morik, K. Joint leaf-refinement and ensemble pruning through l 1 regularization. *Data Mining and Knowledge Discovery*, 37(3):1230–1261, 2023.

[11] Caruana, R., Munson, A., and Niculescu-Mizil, A. Getting the most out of ensemble selection. In *Sixth International Conference on Data Mining (ICDM'06)*, pp. 828–833. IEEE, 2006.

[12] Chaouki, A., Read, J., and Bifet, A. Branches: Efficiently seeking optimal sparse decision trees via ao. In *Forty-second International Conference on Machine Learning*, 2025.

[13] Chen, T. and Guestrin, C. Xgboost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

[14] Cockett, J. Decision expression optimization. *Fundamenta Informaticae*, 10(1):93–114, 1987.

[15] Cockett, J. R. B. Discrete decision theory: Manipulations. *Theoretical Computer Science*, 54 (2-3):215–236, 1987.

[16] Cockett, J. R. B. and Herrera, J. Decision tree reduction. *Journal of the ACM (JACM)*, 37(4): 815–842, 1990.

[17] Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[18] Costa, V. G. and Pedreira, C. E. Recent advances in decision trees: An updated survey. *Artificial Intelligence Review*, 56(5):4765–4800, 2023.

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

[20] Devos, L., Perini, L., Meert, W., and Davis, J. Detecting evasion attacks in deployed tree ensembles. In *Joint European conference on machine learning and knowledge discovery in databases*, pp. 120–136. Springer, 2023.

[21] Donnelly, J., Katta, S., Rudin, C., and Browne, E. P. The Rashomon importance distribution: Getting RID of unstable, single model-based variable importance. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2023.

[22] Donnelly, J., Katta, S., Rudin, C., and Browne, E. P. The Rashomon importance distribution: Getting RID of unstable, single model-based variable importance. In *Advances In Neural Information Processing Systems*, 2023.

[23] Emine, Y., Forel, A., Malek, I., and Vidal, T. Free lunch in the forest: Functionally-identical pruning of boosted tree ensembles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 16488–16495, 2025.

[24] Erickson, N., Purucker, L., Tschalzev, A., Holzmüller, D., Desai, P. M., Salinas, D., and Hutter, F. Tabarena: A living benchmark for machine learning on tabular data. In *Proceedings of the 39th Conference on Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2506.16791.

[25] Ertekin, Ş. and Rudin, C. On equivalence relationships between classification and ranking algorithms. *The Journal of Machine Learning Research*, 12:2905–2929, 2011.

[26] Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. Do we need hundreds of classifiers to solve real world classification problems? *The journal of machine learning research*, 15(1):3133–3181, 2014.

[27] FICO, Google, Imperial College London, MIT, University of Oxford, UC Irvine, and UC Berkeley. Explainable Machine Learning Challenge. https://community.fico.com/s/explainable-machine-learning-challenge, 2018.

[28] Fisher, R. A. Iris. UCI Machine Learning Repository, 1936. DOI: https://doi.org/10.24432/C56C76.

[29] Freund, Y., Schapire, R. E., et al. Experiments with a new boosting algorithm. In *icml*, volume 96, pp. 148–156. Bari, Italy, 1996.

[30] Grinsztajn, L., Oyallon, E., and Varoquaux, G. Why do tree-based models still outperform deep learning on typical tabular data? In *Advances in Neural Information Processing Systems*, volume 35, pp. 507–520, 2022.

[31] Hsu, E., Chen, H., Zhong, C., and Semenova, L. The double-edged nature of the Rashomon set for trustworthy machine learning. *arXiv preprint arXiv:2511.21799*, 2025.

[32] Ignatiev, A., Narodytska, N., and Marques-Silva, J. Abduction-based explanations for machine learning models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1511–1519, 2019.

[33] Izza, Y., Ignatiev, A., and Marques-Silva, J. On tackling explanation redundancy in decision trees. *Journal of Artificial Intelligence Research*, 75:261–321, 2022.

[34] Kantchelian, A., Tygar, J. D., and Joseph, A. Evasion and hardening of tree ensemble classifiers. In *International conference on machine learning*, pp. 2387–2396. PMLR, 2016.

[35] Larson, J., Mattu, S., Kirchner, L., and Angwin, J. How we analyzed the compas recidivism algorithm. *ProPublica*, May 2016. URL https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

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

[37] Lundberg, S. M. and Lee, S.-I. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30, 2017.

[38] Marques-Silva, J. and Ignatiev, A. Efficient & correct predictive equivalence for decision trees, 2025. URL https://arxiv.org/abs/2509.17774.

[39] McTavish, H., Zhong, C., Achermann, R., Karimalis, I., Chen, J., Rudin, C., and Seltzer, M. Fast sparse decision tree optimization via reference ensembles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 9604–9613, 2022.

[40] McTavish, H., Boner, Z., Donnelly, J., Seltzer, M., and Rudin, C. Leveraging predictive equivalence in decision trees, 2025. URL https://arxiv.org/abs/2506.14143.

[41] Nguyen, S., Hoffman, K., and McCormick, T. Unique rashomon sets for robust active learning. *arXiv preprint arXiv:2503.06770*, 2025.

[42] Nguyen, S. D., McTavish, H., Hoffman, K., McCormick, T., and Rudin, C. REALITrees: Rashomon ensemble active learning for interpretable trees. In Perković, E. and Malinsky, D. (eds.), *Proceedings of the 42nd Conference on Uncertainty in Artificial Intelligence*, volume 337 of *Proceedings of Machine Learning Research*, pp. 4832–4854. PMLR, 17–21 Aug 2026. URL https://proceedings.mlr.press/v337/nguyen26a.html.

[43] Quine, W. V. The problem of simplifying truth functions. *The American Mathematical Monthly*, 59(8):521–531, 1952.

[44] Ribeiro, M. T., Singh, S., and Guestrin, C. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[45] Rudin, C., Chen, C., Chen, Z., Huang, H., Semenova, L., and Zhong, C. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistic Surveys*, 16:1–85, 2022.

[46] Rudin, C., Zhong, C., Semenova, L., Seltzer, M., Parr, R., Liu, J., Katta, S., Donnelly, J., Chen, H., and Boner, Z. Amazing things come from having many good models. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

[47] Rudin, C., Zhong, C., Semenova, L., Seltzer, M., Parr, R., Liu, J., Katta, S., Donnelly, J., Chen, H., and Boner, Z. Amazing things come from having many good models. In *International Conference on Machine Learning*, 2024.

[48] Semenova, L., Rudin, C., and Parr, R. On the existence of simpler machine learning models. In *2022 ACM Conference on Fairness, Accountability, and Transparency*, pp. 1827–1858, 2022.

[49] Semenova, L., Chen, H., Parr, R., and Rudin, C. A path to simpler models starts with noise. *Advances in Neural Information Processing Systems*, 36, 2023.

[50] Street, W. N., Wolberg, W. H., and Mangasarian, O. L. Nuclear Feature Extraction for Breast Tumor Diagnosis. In *Biomedical Image Processing and Biomedical Visualization*, volume 1905, pp. 861–870. SPIE, 1993.

[51] Sundararajan, M., Taly, A., and Yan, Q. Axiomatic attribution for deep networks. In *International conference on machine learning*, pp. 3319–3328. PMLR, 2017.

[52] Tollenaar, N. Predicting criminal recidivism using registration data, 2018. URL https://doi.org/10.17026/DANS-23P-7UUU. Dataset Managed by DANS Data Station Physical and Technical Sciences.

[53] Tollenaar, N. and van der Heijden, P. G. Which Method Predicts Recidivism Best?: A Comparison of Statistical, Machine Learning and Data Mining Predictive Models. *Journal of the Royal Statistical Society Series A: Statistics in Society*, 176(2):565–584, 2013.

[54] van der Linden, J., de Weerdt, M., and Demirović, E. Necessary and sufficient conditions for optimal decision trees using dynamic programming. *Advances in Neural Information Processing Systems*, 36:9173–9212, 2023.

[55] Venkateswaran, A., Sankar, A., Chandrasekhar, A. G., and McCormick, T. H. Robustly estimating heterogeneity in factorial data using rashomon partitions. *arXiv preprint arXiv:2404.02141*, 2024.

[56] Verwer, S. and Zhang, Y. Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1625–1632, 2019.

[57] Vidal, T. and Schiffer, M. Born-again tree ensembles. In *International Conference on Machine Learning (ICML)*, pp. 9743–9753. PMLR, 2020.

[58] Wang, C., Han, B., Patel, B., and Rudin, C. In Pursuit of Interpretable, Fair and Accurate Machine Learning for Criminal Recidivism Prediction. *Journal of Quantitative Criminology*, 39(2):519–581, 2023.

[59] Wang, T., Rudin, C., Doshi-Velez, F., Liu, Y., Klampfl, E., and MacNeille, P. A Bayesian Framework for Learning Rule Sets for Interpretable Classification. *Journal of Machine Learning Research*, 18(70):1–37, 2017.

[60] Xin, R., Zhong, C., Chen, Z., Takagi, T., Seltzer, M., and Rudin, C. Exploring the whole Rashomon set of sparse decision trees. In *Advances in Neural Information Processing Systems*, volume 35, pp. 14071–14084, 2022.

[61] Zantema, H., Bodlaender, H. L., et al. Finding small equivalent decision trees is hard. 1999.

[62] Zantema, H. et al. Decision trees: Equivalence and propositional operations. 1998.

A Additional Algorithm Details

A.1 Decision Equivalence Solver

Our decision equivalence algorithm, Algorithm 5, works with a recursive subroutine over subregions, Algorithm 6. The structure is similar to [62], with changes made to remove some other runtime terms aside from the core $\mathcal{O}(|T||T'|)$ asymptotic bound. It works seamlessly with subregion tracking.

Algorithm 5 FINDNONEQUIVALENCE(T,T')

Input: Decision trees T, T'.
Output: If $T(x) \stackrel{\text{DE}}{=} T'(x)$, return None. Else, return a sample x for which $T(x) \neq T'(x)$.
(Let p denote dimensionality of domain of T)
S $\leftarrow (-\infty, \infty)^p$ {Initialize a data structure to track subregion contradictions (initializing with the open interval on all dimensions)}
return SUBREGIONCONTRADICTION(open_region, T, T') {Core logic is in Algorithm 6}

To check pairwise equivalence for baselines, we also use this subroutine (but a variant that does not generate a sample). This variant removes the $\mathcal{O}(p)$ runtime addition (which was only needed for sampling) and runs in $\mathcal{O}(|T||T'|)$ time.

A.2 Categorical Features

The subregions we discuss assume continuous or binary features. Categorical features can be handled similarly for pairwise checks, as queries on whether a single member of a categorical feature is present are only known to be logically impossible if a different query on the same categorical feature is already known to be True, or if every other query on the categorical feature is known to be False (so we just track a count of confirmed false categorical levels relative to all possible; and track if one is known to be True). We focus in the main paper on algorithm details for continuous features, but the extension to categorical is possible with the same algorithms for pairwise checks, we just update subregion tracking as discussed above.

B Theorem Proofs

In this section we show proofs of the claims from the main paper, as well as a few supplementary Lemmas that are used to prove these claims.

Lemma B.1. *For trees T, T' operating over data with p features, Algorithm 5 is correct and has runtime $\mathcal{O}(|T||T'| + p)$.*

Proof. First note that we always make at most one $\mathcal{O}(p)$ call (line 4 in Algorithm 6), which only occurs when we have decided to recurse no further because the trees are proven nonequivalent. It is

Algorithm 6 SUBREGIONCONTRADICTION(S, T, T')

Input: Subregion S, decision trees T, T'.
Output: If $\forall x \in S, T(x) = T'(x)$, return None. Else, return a sample $x \in S$ for which $T(x) \neq T'(x)$.
Invariant: S is modified during execution, but will be in its original state at the end of execution

if T is a leaf **then**
  if T' is a leaf **then**
    if $\hat{y}_T \neq \hat{y}_{T'}$ **then**
      return a sample in S
    else
      return None
    end if
  else
    return SUBREGIONCONTRADICTION(S, T', T) {**Note the swapped order of T' and T**- we now iterate over T', since T has been reduced to a leaf}
  end if
else
  j, t = T.root.query {check query at T's root}
  if $(-\infty, t] \cap S_j \neq \emptyset$ **then**
    old $\leftarrow S_j$
    $S_j \leftarrow S_j \cap (-\infty, t]$
    contradiction = SUBREGIONCONTRADICTION(S, T.left, T')
    $S_j \leftarrow$ old {Restore original S_j value}
    if contradiction $\neq$ None **then**
      return contradiction
    end if
  end if
  if $(t, \infty) \cap S_j \neq \emptyset$ **then**
    old $\leftarrow S_j$
    $S_j \leftarrow S_j \cap (t, \infty)$
    contradiction = SUBREGIONCONTRADICTION(S, T.right, T')
    $S_j \leftarrow$ old
    if contradiction $\neq$ None **then**
      return contradiction
    end if
  end if
end if
Return None

sufficient to show that the decision version of the algorithm (every operation done other than sample generation) runs in $\mathcal{O}(|T||T'|)$ time; the sample generation is then just a single extra $\mathcal{O}(p)$ operation.

Correctness:

The proof will proceed by induction. We will show the subroutine Algorithm 6 correctly evaluates whether $\forall x \in S, T(x) = T'(x)$ (and then it follows trivially that the call on an open interval from Algorithm 5 correctly maps to deciding decision equivalence). We also show the invariant holds (S is the same at the start and end of recursive calls in Alg 6) .

base case: T, T' are both leaves. This part of the subroutine is clearly correct; decision equivalence is an easy problem for this region and we correctly return a sample in the subregion S if the leaves are not equivalent, else None. The invariant on S is maintained because S is not modified.

inductive hypothesis: Let T, T' be two trees and S be the current subregion; assume Algorithm 6 correctly evaluates decision equivalence over all subtrees of T and T' within their corresponding subregions.

inductive step: at least one of T, T' is not a leaf. (Note that if T' is not a leaf and T is, we swap the arguments and continue, with no additional overhead and no further swaps. So we focus here on the case where T is not a leaf). We make two recursive calls on smaller subproblems, checking

if for all x in the subregion where the left path is taken, $T.left(x) = T'(x)$, and if for all x in the subregion where the right path is taken, $T.right(x) = T'(x)$. Since these cases partition the subregion, a positive answer in both cases (by the inductive hypothesis) guarantees the trees are decision equivalent for the current subregion. If either subroutine returns a sample, we propagate that sample correctly (since the sample is also inside this subregion if it was inside the subregion for one of the recursive calls). Note that we restore S_j immediately after the recursive call where we modified the subregion, reversing the operation and restoring the original S_j.

Thus, by induction, the approach is correct.

The runtime is more immediate. We do a standard tree traversal through all leaves of T; for each leaf we do a tree traversal of tree T'. At each split we do a constant number of operations. Thus the runtime is $O(|T||T'|)$, the product of the number of leaves in both trees.

□

Remark B.2. *A variant of Algorithm 5 which returns True for nonequivalence and False for equivalence, without returning a sample, can achieve $\mathcal{O}(|T||T'|)$ complexity.*

Proof. From Lemma B.1, we know the subroutine runs in $\mathcal{O}(|T||T'|)$ except for generating the sample. Here we note simply that the initialization of the subregion can be done in $\mathcal{O}(1)$ time, if we represent the subregion valid intervals with a dictionary/hashmap, and note open intervals as corresponding to a key not existing in the dictionary. □

Proposition 4.1. *Let the original collection be denoted C_d, and let the domain of the trees have dimensionality p. Let the output of Algorithm 1 applied to C_d be C'_d. Then, C'_d satisfies equation (3), and the algorithm has runtime in*

$$\mathcal{O}\left(|C_d|\left(|C'_d|d + \max_{T\in C_d, T'\in C'_d} |T||T'| + p\right)\right).$$

Proof.

Runtime

We will argue that for each tree in C_d, we run a number of tree inferences and PRED_MAP lookups proportional to the size of $X^{(\text{diff})}$ at that iteration, and will run at most one pairwise tree comparison. This will give us the corresponding runtime, for PRED_MAP implemented as a hash table or similar data structure with fast lookups, and noting $|X^{(\text{diff})}| \leq |C'_d|$ throughout the algorithm.

For each tree in C_d, we need to know if any prefix of the tree's predictions on $X^{(\text{diff})}$ is in PRED_MAP (line 4); we can accomplish this by predicting with that tree on each sample in $X^{(\text{diff})}$ in sequence, checking each time for a collision with PRED_MAP until either that collision occurs at index k (making line 4 true) or until we have reached the end of $X^{(\text{diff})}$ with no collisions. In the latter case, we go to line 19, add in the new entry to PRED_MAP, and move on to the next tree, having done only $|X^{(\text{diff})}|$ inferences on the current tree and that many lookups on PRED_MAP.

Now consider the case where line 4 is True. Line 5's value has already been computed by the above process. Now we consider the tree T' which has the same index in PRED_MAP (line 6). To evaluate line 7, we predict with each of the two trees in sequence on $X^{(\text{diff})}$, starting from the (k + 1)th entry in $X^{(\text{diff})}$, and checking if the predictions ever differ before hitting the end of $X^{(\text{diff})}$. If they do, we can safely update PRED_MAP with the two sets of predictions up through this index; in this case we then move on to the next tree in C_d, having in total done no more than 2 tree inferences for each sample in $X^{(\text{diff})}$, along with no more than $X^{(\text{diff})}$ queries on PRED_MAP from the previous step. If they do not collide, then we move on to the else block at line 10.

Consider the case where we have reached the else block on line 10. We know the predictions through all of $X^{(\text{diff})}$, having done no more than 2 tree inferences for each sample in $X^{(\text{diff})}$, along with no more than $X^{(\text{diff})}$ queries on PRED_MAP. We now need to run one instance of the FINDNONEQUIVALENCE algorithm. After that we are either done after adding the colliding tree back into the PRED_MAP with its new index (if the algorithm returns None) or need to run one final inference for each of the two trees on this differing point, and insert both trees back into PRED_MAP. So in total, before

moving to the next tree, we had no more than $|X^{(\text{diff})}|$ inferences (where the size of $X^{(\text{diff})}$ may have increased by 1) for up to two trees, as well as the runtime of FINDNONEQUIVALENCE and no more than $|X^{(\text{diff})}|$ lookups into PRED_MAP.

The worst case of all of the above scenarios is no worse than $|X^{(\text{diff})}|$ lookups into PRED_MAP, each O(1) for a hash table, $|X^{(\text{diff})}|$ inferences for two trees, each bounded by $O(d)$ where d is the maximum depth of the trees, and a call to FINDNONEQUIVALENCE which takes $\mathcal{O}(|T||T'|) + p$ time per Lemma B.1. Considering worst case across the whole outer loop, and noting that $|X^{(\text{diff})}| \leq |C'_d|$ at every step (see the correctness proof below), we have runtime

$$\mathcal{O}\left(|C_d|\left(|C'_d|d + \max_{T \in C_d, T' \in C'_d} |T||T'| + p\right)\right).$$

Correctness

To prove that our collection satisfies the property that all equivalence classes are represented but there are no duplicates, we can use induction over the construction of PRED_MAP and $X^{(\text{diff})}$. We operate here over two key invariants: PRED_MAP contains a single member of all observed equivalence classes, indexed by a prefix of predictions on $X^{(\text{diff})}$ that is unique from all other equivalence classes.

Base case: when we have not yet observed any samples and both PRED_MAP and $X^{(\text{diff})}$ are empty, all invariants hold (vacuously).

Inductive step: If we never hit a collision, then the invariants on PRED_MAP and $X^{(\text{diff})}$ ensure that we have found a new equivalence class, and adding the tree to PRED_MAP indexed with the full prediction vector on $X^{(\text{diff})}$ maintains the invariants (since we haven't hit a collision, we know the prediction vector on $X^{(\text{diff})}$ is unique from any other members of PRED_MAP). If we do hit a collision, then we consider three cases that cover the space of all possibilities: two trees are decision equivalent, they are not decision equivalent and they differ on $X^{(\text{diff})}$, and they are not decision equivalent and they do not differ on $X^{(\text{diff})}$. First, if further predictions on $X^{(\text{diff})}$ for the new tree and the colliding tree eventually cause a difference, we correctly maintain both trees (since they are not equivalent) and index them with a new, unique prefix from each other, maintaining invariants. Second, if the prediction vectors continue to match and we ultimately find the two trees are decision equivalent, we reinsert the old colliding tree with the new index (still maintaining the invariant about unique prefix indices) and correctly do not add the new tree, since it is not a new equivalence class. If the prediction vectors continue to match and the trees are not decision equivalent, then we add a new sample to the end of $X^{(\text{diff})}$ (not violating any previous invariants, since all prefixes of prediction vectors on the old $X^{(\text{diff})}$ are still valid), and this new sample (as per Lemma B.1) corresponds to a sample for which the two trees' predictions differ, so we can update the two prediction vectors and correctly add them, maintaining all invariants. □

Remark 4.2. *Let $R = Rset_\epsilon(\mathcal{D}, \lambda, d)$ be a Rashomon set for some dataset $\mathcal{D}$, slack ϵ, maximum depth d, and leaf penalty $\lambda > 0$. Then its reduced form corresponds to preserving only one tree of lowest objective within each decision equivalence class.*

$$\arg\min_{C'_d \subseteq R} \{|C'_d| \text{ s.t. } \forall T \in R, \exists T' \in C'_d, T' \overset{DE}{=} T, Obj(T', \mathcal{D}, \lambda) \leq Obj(T, \mathcal{D}, \lambda)\} \in \text{REDUCED}(R).$$

Proof. To show this, it is sufficient to show that the decision equivalent form of a tree with the lowest objective maps to its simplest decision equivalent form. (From there, the remainder of the claim is a trivial consequence of the definition of the Rashomon set; all trees of lower objective than a member of the Rashomon set are also within the set, for a given depth d).

Recall Objective 2 used to define the Rashomon set:

$$\text{Obj}(T, \mathcal{D}, \lambda) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{T(x_i) \neq y_i\} + \lambda|T|$$

Without loss of generality, select two trees $T, T' \in \mathcal{T}_d$, $T \overset{\text{DE}}{=} T'$, $\text{Obj}(T', \mathcal{D}, \lambda) \leq \text{Obj}(T, \mathcal{D}, \lambda)$. Then we know that $|T'| \leq |T|$ because the misclassification errors of the two trees must be the same.

More formally:

$$\begin{aligned}
\text{Obj}(T', \mathcal{D}, \lambda) &\leq \text{Obj}(T, \mathcal{D}, \lambda) \\
\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{T'(x_i) \neq y_i\} + \lambda|T'| &\leq \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{T(x_i) \neq y_i\} + \lambda|T| \\
\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{T'(x_i) \neq y_i\} - \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{T(x_i) \neq y_i\} + \lambda|T'| &\leq \lambda|T| \\
\frac{1}{n}\sum_{i=1}^{n} (\mathbb{1}\{T'(x_i) \neq y_i\} - \mathbb{1}\{T(x_i) \neq y_i\}) + \lambda|T'| &\leq \lambda|T| \\
\frac{1}{n}\sum_{i=1}^{n} (\mathbb{1}\{T(x_i) \neq y_i\} - \mathbb{1}\{T(x_i) \neq y_i\}) + \lambda|T'| &\leq \lambda|T| \quad \text{(By decision equivalence)} \\
\lambda|T'| &\leq \lambda|T| \\
|T'| &\leq |T|
\end{aligned}$$

As required.

□

Lemma 4.3. *Let $\mathcal{D}_{missing} = \{(m_{i'}, y_{i'})\}_{i'=1}^{|T'|}$ be the output of Algorithm 2 for tree T'. If the worst-case classification error of tree T across any completion is 0, then $T \overset{DE}{=} T'$. If no path in T or T' is impossible (contains a query with value already known from earlier queries along that path), and queries are all of the form $\mathbb{1}\{x_{ij} \leq t\}$, then this becomes bidirectional:*

$$T \overset{DE}{=} T' \iff \left(\sum_{i'=1}^{|T'|} \max_{x_{i'} \in Completions(m_{i'})} \mathbb{1}\{T(x_{i'}) \neq y_{i'}\} \right) = 0$$

Proof. We begin by showing that **0 error implies decision equivalence**.

Note that for a dataset containing every possible point in the domain $\mathcal{X}$, with labels given by T''s predictions, 0 error on this dataset would yield a decision equivalent tree T by definition.

Since leaves partition the domain, any possible point in the domain corresponds to a completion of a row in $\mathcal{D}_{missing}$.

Since a tree's leaves are implicants for that tree's prediction, the labels for any completion of a row in $\mathcal{D}_{missing}$ will be consistent with T''s classification of that completion. This is a consequence of the definition of completions and the fact that each row maps to a leaf implicant for tree T': sample $x \in \mathcal{X}$ corresponds to a completion of row i of $\mathcal{D}_{missing}$ if and only if x is routed to leaf ℓ_i in $leaves(T')$.

Therefore, if some tree T has 0 error over any possible completion of any row in $\mathcal{D}_{missing}$, it must match tree T''s predictions over any possible point in the domain. Then these two trees must be decision equivalent by definition.

We now show that **decision equivalence implies 0 error**.

For most completions of the leaf dataset, this follows immediately from the definition of decision equivalence. We know from the definition of decision equivalence that for all realizable samples in the domain $\mathcal{X}$, two decision equivalent trees will classify those samples the same way. And we know the labels correspond to the predictions of T' for all samples, because of how we have defined the missingness dataset.

It is possible, however, for there to be some completion of the dataset that cannot be realized by any sample in the domain $\mathcal{X}$ (see Definition B.4).

Definition B.3 (Inconsistent assignments)**.** *Consider a set of splits of the form $\mathbb{1}[v_j \leq t]$. Define an inconsistent assignment as an assignment of values to two splits that no $v \in \mathcal{X}$ can simultaneously*

satisfy. That is, for two thresholds such that $t_1 < t_2$, any assignment such that $\mathbb{1}[v_j \leq t_1] = 1 \wedge \mathbb{1}[v_j \leq t_2] = 0$. A path that requires such an inconsistent assignment to traverse is an impossible path.

Definition B.4 (Impossible path)**.** *A path in a tree that requires an inconsistent assignment to traverse is an impossible path.*

To complete our proof, we need to show that two decision equivalent trees T and T' are also equivalent on any row completions of $\mathcal{D}_{missing}$ that correspond to inconsistent assignments. We handle this case with the following lemma:

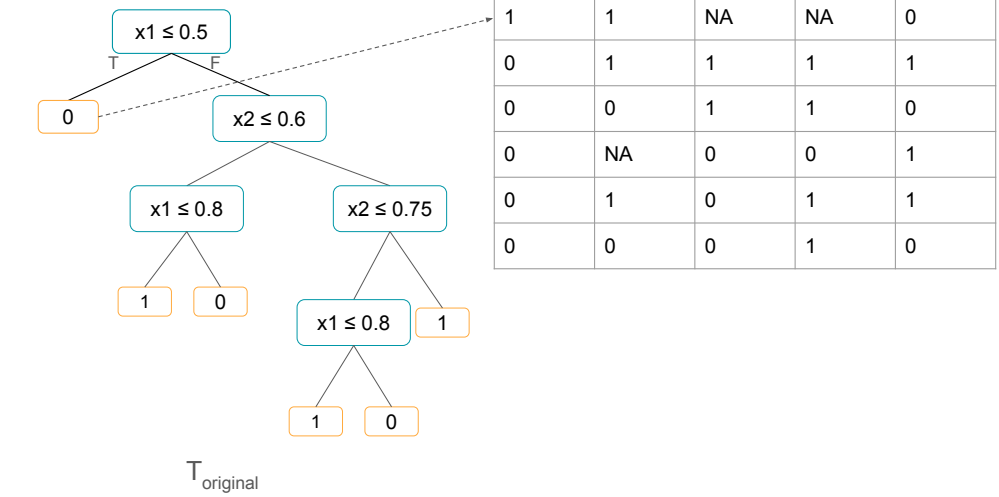
Lemma B.5. *Let T' and T be trees over the domain $\mathcal{X}$ such that neither has impossible paths. Define $\mathcal{D}_{missing}$ to be the output of Algorithm 2 for T'. For any completion of $\mathcal{D}_{missing}$ containing an inconsistent assignment, there exists some other, in-domain completion of the same row of $\mathcal{D}_{missing}$ that is classified identically by T.*

Proof. Consider, without loss of generality, any row r of $\mathcal{D}_{missing}$. First note a simple property of Algorithm 2: any completion of r must be consistent with all known query values in r. This follows from the processing in Algorithm 2, which only maintains NA in a column corresponding to $\mathbb{1}[v_j \leq t]$ if both true and false values are possible given the known query values in that row.

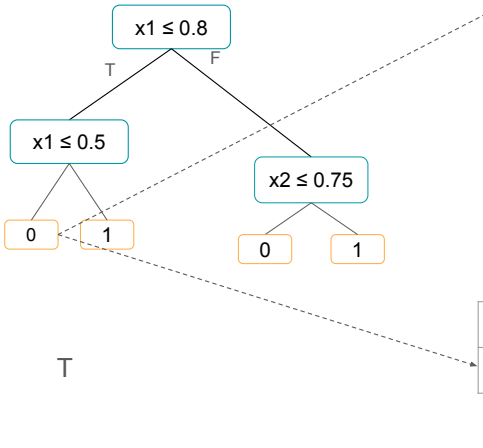
Now, consider any completion of this row containing an inconsistent assignment, and consider the leaf it reaches in T (call it ℓ_T. We know the inconsistent assignment cannot involve values fixed for r, by the argument above. And the inconsistent assignment cannot be solely among values necessary to reach this leaf ℓ_T in T – that is, there must be at least one split involved in the inconsistent assignment that is not queried along the path to leaf ℓ_T – because T contains no impossible paths. Thus the inconsistencies must involve some queries that are not necessary for reaching leaf ℓ_T and are not fixed for r (that is, they are NA in r). Since these values are NA, all possible assignments of these values are valid completions consistent with known values in r. We also know at least one valid set of assignments of these values will be consistent with reaching the leaf ℓ_T, since no path in T is impossible. Thus, we can construct another completion of row r that is consistent and reaches the same leaf. □

With this lemma, we can complete the proof. Decision equivalence means all in-domain samples are classified identically for T and T'. Now, for any out-of-domain sample, it must be classified the same way by T as an in-domain sample completing the same row by Lemma B.5. Since both are completions of the same row, T must also classify them correctly. Thus, T will achieve 0 error, completing the proof.

□



x1 ≤ .5	x1 ≤ .8	x2 ≤ .6	x2 ≤ .75	$\hat{y}_{tree}$
1	1	NA	NA	0
0	1	1	1	1
0	0	1	1	0
0	NA	0	0	1
0	1	0	1	1
0	0	0	1	0



x1 ≤ .5	x1 ≤ .8	x2 ≤ .6	x2 ≤ .75	$\hat{y}_{tree}$
1	1	1	0	0

impossible completion

x1 ≤ .5	x1 ≤ .8	x2 ≤ .6	x2 ≤ .75	$\hat{y}_{tree}$
1	1	0	0	0

possible completion of same row

Figure 2: Visualization of one example of matchings from lemma B.5.

Lemma 4.4. *Let $T^{mod}_{\mathcal{D},d,\lambda}$ be a tree output from an optimal decision tree solver modified with Algorithm 3. Instead of being optimal for objective (1), this tree is optimal for an upper bound on Equation (1) for the worst-case completion of a dataset, subject to a sparsity penalty:*

$$T^{mod}_{\mathcal{D},d,\lambda} \in \underset{T\in\mathcal{T}_d}{\arg\min} \left(\lambda|T| + \sum_{\ell\in leaves(T)} \max_{\{(x_{i'},y_{i'})\}_{i'=1}^{|\mathcal{D}|}\in Completions(\mathcal{D})} \sum_{i'=1}^{|\mathcal{D}|} \mathbb{1}\{x_{i'} \in S(\ell) \wedge y_{i'} \neq \hat{y}_\ell\} \right)$$

Proof. Note that recursive dynamic programming optimal tree solvers like STreeD work by recursively partitioning the data into left and right subregions, deciding on a tree making a split which corresponds to minimal error in these subregions (and filling in with optimal subtrees for these subregions); or making a leaf prediction if that is better.

The modification in Algorithm 3 guarantees that as we consider splits, for each row in $\mathcal{D}$ where there exists a completion that is consistent with the subregion, the matched subregion will contain that row. This follows fairly directly: we can only remove a row from a subregion if it corresponds to the False (resp. True) path of a feature that is True (resp. False) and not NA (that is, the row is forced to conflict with that path no matter what). Otherwise, there exists at least one completion that is consistent with the path (either all completions are consistent, or we can fix an NA value to match that query).

As such, we know that the number of errors a tree makes in any subregion where it predicts a leaf is at least the number of rows with completions that conflict with the leaf prediction. This follows because we include any row that has a conflicting label for at least one completion within that subregion. Assuming an optimizer that fixes n in its objective calculation as the size of the original dataset (such an implementation detail is trivial to guarantee), that means minimizing misclassifications across all these subregions corresponds to minimizing an upper bound on the worst case completion of a dataset. □

Theorem 4.5. *Algorithm 4 returns a sparsest decision equivalent form of the input tree within the specified depth bound, if one exists.*

Proof. This follows from Lemmas 4.3 and 4.4. We do need a few extra details. First, note that we set the lambda value so low that the minima returned by Algorithm 4 must make 0 errors. Because we have set lambda so low that its contribution to the objective on T' is less than 1 misclassification, but the lambda is still nonzero, Algorithm 4 must find the sparsest form achieving 0 error on this upper bound. Second, note that 0 error for the summation in Lemma 4.4 can occur if and only if there is 0 error for the summation in Lemma 4.3. If we had a nonzero error in either, it means we could find a nonzero error in the other.

Thus we have made a dataset such that 0 error across all completions occurs if and only if the found tree is decision equivalent to the original tree. We then picked a hyperparameter setting for lambda such that this algorithm must achieve 0 error in the sparsest form possible.

Therefore, if a sparsest decision equivalent form exists, it will be returned. □

Remark 4.6. *Consider any ensemble of m binary leaf decision trees $C = [T^{(1)}, \dots T^{(m)}] \in \mathcal{T}_d^m$ and weights $[w^{(1)}, \dots, w^{(m)}]$. If the ensemble's probability prediction is*

$$\hat{p}(x) = \frac{1}{\sum_{i=1}^m w^{(i)}} \sum_{i=1}^m w^{(i)} T^{(i)}(x),$$

Then its predicted probabilities are exactly equivalent for the alternative ensemble $[T'^{(1)}, \dots T'^{(k)}] =$ REDUCED(C), with weights $[w'^{(1)}, \dots, w'^{(k)}]$ such that $w'^{(j)} = \sum_{i=1}^m w^{(i)} \mathbb{1}\{T^{(i)} \overset{DE}{=} T'^{(j)}\}$.

Proof. For simplicity of notation, denote $\bar{w}^{(i)} := \frac{w^{(i)}}{\sum_{i=1}^m w^{(i)}}$, giving

$$\hat{p}(x) = \sum_{i=1}^m \bar{w}^{(i)} T^{(i)}(x)$$

Since we know that $C' \in \text{REDUCED}(C)$, from Definition 3.3, we know that each tree $T^{(i)} \in C$ is decision equivalent to exactly one tree $T'^{(j)}$ in the new collection. So, we have that:

$$\begin{aligned}
\hat{p}(x) &= \sum_{i=1}^{m} \bar{w}^{(i)} T^{(i)}(x) \\
&= \sum_{i=1}^{m} \left(\bar{w}^{(i)} T^{(i)}(x) \sum_{j=1}^{k} \mathbb{1}\{T^{(i)} \overset{\text{DE}}{=} T'^{(j)}\} \right) && \text{Multiply by 1} \\
&= \sum_{j=1}^{k} \sum_{i=1}^{m} \left(\bar{w}^{(i)} T^{(i)}(x) \mathbb{1}\{T^{(i)} \overset{\text{DE}}{=} T'^{(j)}\} \right) && \text{Swap summation order} \\
&= \sum_{j=1}^{k} \sum_{i=1}^{m} \left(\bar{w}^{(i)} T'^{(j)}(x) \mathbb{1}\{T^{(i)} \overset{\text{DE}}{=} T'^{(j)}\} \right) && \text{By definition of } \overset{\text{DE}}{=} \\
&= \sum_{j=1}^{k} \sum_{i=1}^{m} \left(\bar{w}^{(i)} \mathbb{1}\{T^{(i)} \overset{\text{DE}}{=} T'^{(j)}\} \right) T'^{(j)}(x) \\
&= \sum_{j=1}^{k} \bar{w}'^{(j)} T'^{(j)}(x),
\end{aligned}$$

where $\bar{w}'^{(j)} := \sum_{i=1}^{m} \left(\bar{w}^{(i)} \mathbb{1}\{T^{(i)} \overset{\text{DE}}{=} T'^{(j)}\} \right)$. Thus, we have that $\hat{p}(x) = \sum_{j=1}^{k} \bar{w}'^{(j)} T'^{(j)}(x)$ as required. □

C Further Experimental Details

C.1 Additional Results

Here we tune Adaboost on the following full grid using 5 fold cross validation. 'n_estimators': [50, 100, 1000],'learning_rate': [0.01, 0.1, 1], decision tree depth 1, 2, 3, 4, 5, no limit. We then compare to forcing 100 depth 3 estimators and only tuning the learning rate on the same values of [0.01, 0.1, 1]. Note that there is comparable AUC in both cases; we take this as evidence that the configuration restriction we use in the main paper is a reasonable restriction for AdaBoost.

Dataset	AdaBoost	AdaBoost3
Iris_setosa	1.000 ± 0.000	1.000 ± 0.000
Iris_virginica	0.973 ± 0.011	0.960 ± 0.017
Iris_versicolor	0.968 ± 0.018	0.989 ± 0.006
Spiral	0.691 ± 0.089	0.756 ± 0.062
Tic-Tac-Toe	0.998 ± 0.001	1.000 ± 0.000
Broward	0.650 ± 0.004	0.649 ± 0.006
COMPAS	0.734 ± 0.007	0.734 ± 0.007
Wisconsin	0.994 ± 0.003	0.994 ± 0.002
FICO	0.743 ± 0.011	0.745 ± 0.010
Wine Quality	0.819 ± 0.004	0.817 ± 0.005
Coupon	0.764 ± 0.004	0.760 ± 0.003
Netherlands	0.768 ± 0.002	0.764 ± 0.001

Table 4: Test AUC when tuning a grid of parameter values for Adaboost, is comparable to that when constraining the grid to 100 depth 3 trees. Error bars are standard error for 5 outer folds (doing an inner 5 fold CV search for each outer fold).

C.2 Computational Details

Timing experiments for AdaBoost use Dell R730 Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.4GHz,25M Cache (256GB RAM - 40 cores), with 50GB requested; other AdaBoost experiments use similar

computing power. For the Rashomon set timing experiments with 100GB we use Exxact TS4-130227231 with AMD EPYC 9554 64-Core Processor (1TB RAM - 64 cores, 128 threads). Other experiments less sensitive to timing or hardware used whichever machines were available on our department cluster, with similar or less computational power. We suspect that most experiments except for the largest Rashomon sets run can be done quite feasibly with less; absolute timings will change but relative timing conclusions should be roughly consistent with the results from this paper.

C.3 Datasets and Assets

We consider the 12 main datasets used in [40] which we refer to as Netherlands [52, 53], Broward [58], FICO [27], Spiral [39], Tic-Tac-Toe [2], and Iris Setosa/Versicolor/Virginica [28], as well as COMPAS [35], Wine Quality [17], Wisconsin [50], and Coupon [59]. COMPAS measures 7 features for 6,907 individuals, where labels are whether the individuals were arrested within 2 years of being released from prison. Wine Quality reports 11 features over 6,497 wines along with a numerical quality rating between 1 and 10. We binarize these ratings into high (> 5) and low (≤ 5) quality classes and predict this binary rating. Wisconsin is a breast cancer dataset and contains 30 features over 569 masses, where labels designate whether the tumor was malignant or benign. Coupon measures 25 features for 12,684 individuals, and labels denote whether or not the individual would accept a coupon. Netherlands measures 9 features for 20,000 individuals, where labels are whether the individuals from the Netherlands committed another crime after being released from prison. Similarly, Broward reports 38 features from 1,955 individuals from Broward county, Florida, where labels are whether the individuals from the committed another crime after being released from prison. FICO contains 23 features from 10,459 individuals, and labels whether each individual will repay a line of credit within 2 years. In our experiments, we remove all rows from FICO that contain missing data, resulting in 2,502 samples. Spiral is a synthetic dataset with 2 features over 100 samples, where each sample is randomly drawn from one of two interweaving spirals and the features are the x and y coordinates of the sample. In spiral, labels indicate which spiral each sample was drawn from. Tic-Tac-Toe measures 9 features over all 958 possible games of tic-tac-toe, and labels each game as 1 if the first player won and 0 otherwise. The three Iris datasets – Iris Setosa, Iris Versicolor, and Iris Virginica – are different tasks generated from the same multiclass classification dataset. The original Iris dataset reports 4 features over 150 iris flowers, and labels each sample as Setosa, Versicolor, or Virginica. We transform this dataset into three one versus all binary classification datasets.

Most datasets are from UCI, with a creative commons license. The license for FICO could previously be found at `https://community.fico.com/s/explainable-machine-learning-challenge?tabset-158d9=2`, although this link is no longer valid as of the camera-ready date. The license for Netherlands is available from contacting the authors of the paper [53], with the dataset managed by DANS Data Station Physical and Technical Sciences (i.e., the managing entity for the formal dataset citation [52]). Spiral is a synthetic dataset from [39].

We also make use of the codebase from McTavish et al. [40] in our experimental comparisons and in some of the implementation of our own method; that codebase has an MIT license.

C.4 Preprocessing and further details

To match prior work, in binarizing for Rashomon sets we used the internal binarization procedure from [22, 40], which binarizes according to Threshold Guessing from [39], using the thresholds selected based on 40 boosted decision stumps. We do not binarize for the Adaboost experiments, though we note for future work that quantizing could help with instances where there are two nearly identical splits in a tree.

For quantifying the amount of compression our Rashomon sets match the parameters given in [40] for related experiments, and use depth 3 (i.e., the $depth_bound$ parameter was set to 4), additive Rashomon bound 0.02, and sparsity penalty 0.01.

D Properties Preserved Within a Decision Equivalence Class

This section includes a brief set of remarks about which metrics are preserved when we change a tree within its decision equivalence class.

Broadly speaking, any property that depends solely on the classifier's prediction (on any possible sample, whether or not it is in-distribution) will be preserved. Properties that depend on the evaluation order implied by the tree, independently of the classifier's prediction, may not.

For variable importance or feature attribution, remaining in a decision equivalence class for a single tree will not change any variable importance metric that obeys standard axioms [37, 51]. Some variable importance metrics, such as Gini importance, do not obey this property (as shown in [40]); we view this as a flaw in such variable importance metrics, rather than a limitation of changing a tree within an equivalence class. Another property that is not preserved is the presence of a meaningless feature - that is, a feature which, no matter what it is changed to, cannot influence predictions. All such splits are removed in the sparsest equivalent form of a tree.

In a setting where each feature maps to a cost (for example, different tests one might administer in a medical setting), selecting the sparsest equivalent form may not minimize the expected cost to evaluate the tree, or the most equitable partitioning of expected cost across subgroups.

We also note that one promising direction of future work – to infer feature dependencies from data and define equivalence only within the set of points that are possible according to such a model – can introduce complications that are not present in this work. Consider a biased dataset where some grouping of variables is a proxy for a protected attribute. In our method, we can never find a sparsest form that reasons directly on the protected attribute when given a tree that reasons only on the proxy. However, a method that infers feature dependencies may make this simplification based on the observed data. While we still view this direction of future work as promising, this risk would be a relevant consideration for such work.

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: The abstract clearly depicts the contributions and scope: we focus on collections of trees with binary leaves, and explore compression of Rashomon sets and AdaBoost classifiers.

   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 discuss limitations in the conclusion, primarily angled as future work: the NP hard sparsest form solver means it is an excellent direction for future work to extend from this exact shallow approach to handling deeper trees with improvements to algorithm efficiency or with heuristic approaches not guaranteed a sparsest form for each tree, but still seeking a sparser one. We also mention that it would be excellent for future work to relax the restriction to binary leaves.

   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: We provide proofs and assumptions in the appendix.

   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 details in the appendix on compute and describe our algorithms in the main paper and appendix.

   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: Code is included, as well as most data (bar the Netherlands dataset, which we do not have permission to distribute).

   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 provide these details in the experiments section and appendix.

   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: Error bars are denoted as standard error across 5 folds.

   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: This is included in the appendix.

   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 conformed with the Code of Ethics.

   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: This paper provides a general method without a 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: There are no high risk data used in this paper.

    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 provide this in the appendix. Note that the FICO dataset has a license link that, while previously accessible, is no longer accessible to the best of our knowledge as of this paper's camera ready deadline; it remains available on OpenML, was the subject of a public competition, and is a regular point of study for benchmarking and academic papers [24].

    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: Supplementary code is also released along with this paper, with comments about replication. Datasets and models are documented in the paper.

    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 conducted no human subjects research in this work.

    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: We conducted no human subjects research for this work.

    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: We did not use LLMs as part of the core methods in this research.

    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.