PRIOR-GUIDED GLOBAL SEARCH BEYOND GREEDY SPLITTING FOR INTERPRETABLE DECISION TREES

Anonymous authors
Paper under double-blind review

ABSTRACT

Decision trees remain widely used in regulated settings because each prediction traces back to an explicit rule. However, the standard greedy top-down induction used by CART commits to one split at a time and therefore misses patterns that only pay off after several coordinated splits. Exact solvers such as GOSDT and DL8.5 remove this myopia, but they rely on binarized features and frequently exhaust memory or time on wider datasets. We present PG-DTS, which casts tree induction as a deterministic Markov decision process and searches over whole tree structures under a fixed compute budget, while the output remains an ordinary axis-aligned decision tree. Split rewards are computed from the affected leaf alone and cached in a Global Leaf Registry, a PUCT prior steers the budget toward promising splits, and transposed states are deduplicated while tree materialization is deferred until needed. On an XOR trap where CART stalls at 53.72% accuracy, PG-DTS reaches 70.42%, matching the exact solvers, and under 20% label noise it is the most accurate method in every synthetic scenario. On 12 OpenML datasets at depth 4, it achieves the highest accuracy on 7 and returns a tree on all 12, whereas GOSDT fails on 3 and removing the leaf registry roughly quadruples runtime. These results suggest that prior-guided global search can reduce the myopia of axis-aligned trees while keeping their rules explicit.

1 INTRODUCTION

Despite the exceptional performance of deep neural networks (He et al., 2016), their inherent black-box nature restricts their deployment in high-stakes domains such as medical diagnosis and quantitative finance (Rudin, 2019; Arrieta et al., 2020). In strictly regulated scenarios such as those shaped by the General Data Protection Regulation (GDPR) (Goodman & Flaxman, 2017), Decision Trees (DTs) remain a foundational paradigm owing to their transparent and intuitive rule structures (Costa & Pedreira, 2023). However, mainstream top-down greedy induction methods, including Classification and Regression Trees (CART) (Breiman et al., 1984), lack a global perspective. When encountering complex feature interactions like XOR logic, they often converge to sub-optimal, structurally redundant solutions (Minsky & Papert, 1969).

To overcome such local optima, recent exact global solvers, including GOSDT and DL8.5, provide rigorous optimality guarantees. However, they heavily rely on data binarization, frequently encountering memory (OOM) and computational bottlenecks on high-dimensional datasets (Bertsimas & Dunn, 2017; Aglin et al., 2020; Lin et al., 2020). Heuristic search techniques, such as Monte Carlo Tree Search (MCTS), can balance global exploration with computational overhead (Kocsis & Szepesvári, 2006; Nunes et al., 2020). Nevertheless, applying MCTS to offline, full-batch data still faces three major challenges: (1) Without prior guidance, blindly searching an action space that grows exponentially with tree depth is highly inefficient; (2) Traditional whole-tree accuracy evaluation requires top-down data inference for every node expansion, incurring prohibitive overhead; (3) Permuting feature split sequences produces "state transpositions," causing severe memory waste and redundant computations (Childs et al., 2008).

Addressing these challenges, we propose PG-DTS, a Prior-Guided Global Decision Tree Search framework that moves beyond greedy split selection while retaining the transparency of standard decision trees. Rather than committing irrevocably to locally optimal splits, PG-DTS formulates tree induction as a deterministic Markov Decision Process (MDP) and performs global search over tree

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

structures under a fixed computational budget. This design positions PG-DTS as a practical middle ground between fast but myopic greedy induction and globally optimal but often brittle exact solvers. Our core contributions are:

- **Global search for intrinsically interpretable trees:** We formulate axis-aligned decision tree induction as a deterministic MDP and use prior-guided tree search to escape greedy local optima while preserving compact, human-readable decision rules.
- **Efficient leaf-level reward reuse:** We introduce a locality-based reward decomposition and a Global Leaf Registry (GLR), allowing split rewards to be evaluated using only the affected leaf rather than repeatedly performing full-tree inference.
- **Scalable prior-guided search under real-world constraints:** We combine PUCT-based heuristic priors, transposition-aware state deduplication, and lazy tree instantiation to reduce redundant exploration and memory overhead, enabling global search on high-dimensional tabular datasets.

2 Related work

To address local optima in decision tree induction, various optimization directions have been proposed. This section reviews existing construction methods and analyzes their performance and limitations in real-world scenarios.

2.1 Local heuristic induction and exact global solvers

Classic decision tree induction relies on local heuristic induction, including ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993), and CART (Breiman et al., 1984). These algorithms greedily select features maximizing immediate purity gains. While computationally efficient, the lack of backtracking mechanisms makes them susceptible to sub-optimal convergence and overfitting on noisy features (Barros et al., 2011). To seek global optimality, researchers have developed Exact Global Solvers. Representative works include OCT based on mixed-integer optimization (Bertsimas & Dunn, 2017), as well as DL8.5 (Aglin et al., 2020) and GOSDT (Lin et al., 2020) utilizing dynamic programming and branch-and-bound. Theoretically, these solvers guarantee optimal trees; however, their heavy reliance on pre-binarization frequently triggers feature space explosion and memory (OOM) bottlenecks on large-scale datasets (Lin et al., 2020).

2.2 Tree search based on RL and MDP

With the advancement of Reinforcement Learning (RL), recent literature has integrated Markov Decision Processes (MDPs) into decision tree optimization. Frameworks such as DPDT (Kohler et al., 2025) rigorously formalize tree induction as an MDP, utilizing dynamic programming and heuristic split generation to search for near-optimal structures. Complementarily, the TAO algorithm (Carreira-Perpiñán & Tavallali, 2018) employs an alternating optimization strategy to iteratively refine node parameters within a given tree structure. TAO is particularly well-suited for learning sparse oblique trees, where decision nodes utilize linear combinations of features to model correlated data effectively (Carreira-Perpiñán & Tavallali, 2018). In contrast, this paper targets hard decision trees with univariate axis-aligned splits, which navigate a fundamentally different action and hypothesis space.

2.3 Applications and limitations of MCTS in decision tree construction

As a vital branch of heuristic planning, Monte Carlo Tree Search (MCTS) balances global optimization accuracy with computational overhead (Kocsis & Szepesvári, 2006). While Nunes et al. (2020) pioneered an early MCTS framework for decision tree optimization, its prohibitive computational cost stems from the reliance on greedy tree completion followed by top-down re-inference on full datasets. For online streaming, Chaouki et al. (2024) recently proposed the TSDT algorithm using Thompson Sampling. While effective for online data randomness, maintaining posterior distributions is less natural for the deterministic offline batch setting considered here. Existing MCTS-based tree induction methods also provide limited support for exploration priors and for managing equivalent

states (Childs et al., 2008). PG-DTS addresses this gap by moving reward evaluation to the leaf level, sharing leaf statistics through a Global Leaf Registry, and combining prior-guided exploration with state deduplication and lazy instantiation.

3 PROBLEM FORMULATION

This section first rigorously formalizes the top-down generation process of a decision tree as a Markov Decision Process (MDP) and proposes a locality-based reward function at the leaf-node level. Subsequently, based on this property, it introduces a global information-sharing mechanism for leaf nodes.

3.1 DETERMINISTIC MDP FORMULATION

Given an input space $\mathbf{X}$ containing p-dimensional features and target labels Y, the inference process of a decision tree essentially partitions the input space into mutually exclusive sub-regions (corresponding to leaf nodes $l \in \mathcal{L}$). We model this iterative partitioning process as a deterministic Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$:

- **State Space $\mathcal{S}$:** Each state $s \in \mathcal{S}$ represents a partially constructed decision tree with a specific structure. The initial state s_{root} is a single root-node tree containing the entire training dataset.
- **Action Space $\mathcal{A}$:** Under state s, the action space $\mathcal{A}(s)$ is defined as the set of valid operations for binary splitting specific leaf nodes. Specifically, an action $a = (l, X_i, th)$ represents selecting a leaf node $l \in \mathcal{L}(s)$ and utilizing feature X_i with threshold th to strictly partition its data into two mutually exclusive subsets ($X_i \leq th$ and $X_i > th$).
- **Transition Function $\mathcal{P}$:** The decision tree construction environment is entirely deterministic. Executing a valid split action a in state s inevitably generates a unique subsequent state s' with a newly partitioned structure (i.e., $\mathcal{P}(s'|s, a) = 1$).

The search procedure that optimizes this MDP is presented in Section 4, and the pseudocode of all its phases is given in Appendix A.

3.2 LOCALITY-BASED REWARD DECONSTRUCTION

The global accuracy $H(DT)$ of a decision tree decomposes into the weighted sum of its leaf purities: $H(DT) = \sum_{l \in \mathcal{L}(DT)} p(l)H(l)$, where $p(l)$ is the sample proportion and $H(l)$ is the majority class accuracy. We define the single-step reward as the increment in global accuracy: $r(s_t, a_t) = H(s_{t+1}) - H(s_t)$.

When action a_t splits a target leaf l into $\{l_{\text{left}}, l_{\text{right}}\}$, the contributions of all unaltered leaves cancel out. Consequently, the reward is minimally deconstructed as:

$$r(s_t, a_t) = p(l_{\text{left}})H(l_{\text{left}}) + p(l_{\text{right}})H(l_{\text{right}}) - p(l)H(l). \tag{1}$$

This locality-based formulation avoids repeated full-tree forward inference when evaluating a candidate split, since only the affected leaf needs to be updated.

Coupled with an undiscounted factor ($\gamma = 1$), the cumulative reward forms a telescoping series. Thus, maximizing the MDP value function $V^{\pi}(s_{\text{root}})$ is mathematically equivalent to finding the optimal tree with the highest global accuracy (detailed proof is provided in Appendix B).

3.3 GLOBAL LEAF INFORMATION SHARING

Based on the aforementioned reward function, we designed a Global Leaf Registry (GLR) to facilitate information sharing across search branches. In decision tree construction, a data subset within a leaf node is uniquely and entirely determined by the top-down partition constraints (e.g., $X_1 \leq th_1 \wedge X_2 > th_2$). Decision trees corresponding to different search nodes may contain leaf nodes with identical data subsets. Therefore, we extract the unordered hash value of this constraint set as

the Key to permanently cache various statistics (sample count, class distribution, purity, etc.) of this data subset in the GLR.

When MCTS evaluates a split whose resulting leaf constraints have appeared in another search branch, the algorithm can reuse the cached statistics through an $\mathcal{O}(1)$ hash query instead of scanning the data again. We further use the complementarity of binary splits through a "compute one side, deduce the other" rule. If neither child is cached, PG-DTS computes the statistics for one child and derives the other by subtracting the known child statistics from the parent leaf. This reduces repeated data slicing across search branches while keeping the reward computation tied to the same leaf-level objective.

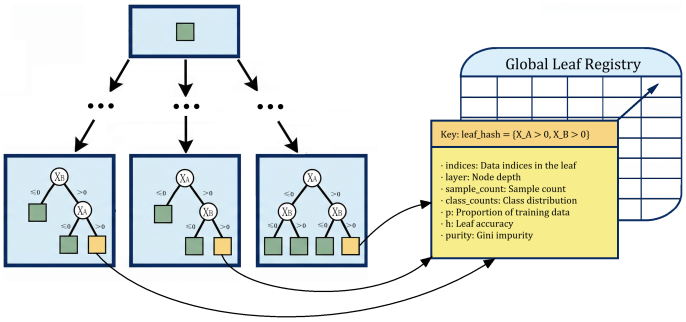


Figure 1: Illustration of the Global Leaf Registry (GLR) mechanism for information sharing across search branches.

4 Proposed Method

This section elaborates on the internal mechanisms of PG-DTS, the Prior-Guided Global Decision Tree Search framework. PG-DTS integrates the classic MCTS paradigm with heuristic optimization. Building upon the local evaluation mechanism established in Section 3, it further introduces an exploration prior, alongside search-node-based state deduplication and lazy-build mechanisms.

Before detailing the search procedure, we clarify three terms that are used throughout this section. An *action recipe* is the triple (l, X_i, th) that fully determines one split, namely the target leaf l, the split feature X_i, and the threshold th. *Lazy-Build* denotes the mechanism by which a child search node is created without materialising its decision tree: only the action recipe is stored, and the tree is instantiated once that node is selected for further exploration or chosen when extracting the final model. The *PUCT strategy* is the rule applied in the Select phase to choose among the children of a node, combining exploitation of the expected value Q with exploration scaled by the prior P and the visit counts.

4.1 Overall Architecture

Within a given computational budget M, the search process refines the value estimate of the root node through four phases: Select, Expand, Simulate, and Backup. After the budget is exhausted, PG-DTS extracts the best tree found by following the path with the highest expected value Q and uses the corresponding decision tree for prediction. The depth constraint enters through the Expand phase: a leaf is no longer expanded once it reaches the maximum depth d, once it is pure, once it contains too few samples (less than 0.1% of the training set), or once no valid split action remains for it. Every split is therefore a binary split of an individual leaf, and the resulting tree never exceeds depth d.

Select: Starting from the root node, the algorithm traverses down the search tree utilizing the PUCT strategy infused with heuristic priors until it locates a boundary node with exploration and expansion potential. During this downward traversal, if the selected child node only contains an action recipe, the Lazy-Build mechanism is triggered, instantly instantiating its internal decision tree using its parent node's state.

Expand: If the selected boundary node is not fully expanded, a candidate action set $\{(l, X_i, th)\}$ is generated. For each feature, the candidate thresholds are its unique discretised values except

Algorithm 1 PG-DTS Main Search Algorithm

Input: Training data $\mathcal{D}$, Search iteration budget M
Output: Optimal decision tree model T^*
Initialize root node s_{root} using data $\mathcal{D}$
$\mathcal{T} \leftarrow \emptyset$, $\mathcal{L} \leftarrow$ INITIALLEAVES($\mathcal{D}$) ▷ Initialize Transposition Table and Leaf Registry
for $i = 1$ **to** M **do**
  $s \leftarrow$ SELECT(s_{root}) ▷ Select boundary node
  if not s.is_terminal **and not** s.is_expand **then**
    EXPAND($s, \mathcal{T}, \mathcal{L}$) ▷ Expand non-terminal node
  end if
  BACKUP(s) ▷ Backpropagate to update values
end for
$s \leftarrow s_{\text{root}}$
while s.children $\neq \emptyset$ **do**
  $s \leftarrow \text{argmax}_{s' \in s.\text{children}} (s'.Q)$ ▷ Extract path with max expected value
  if s.dt_state == NULL **then**
    $(l, X_i, th) \leftarrow s$.split_recipe
    s.dt_state $\leftarrow$ PHYSICALSPLIT(s.parent.dt_state, l, X_i, th)
  end if
end while
return s.dt_state

the largest one, restricted to those that fall between the minimum and maximum values observed inside the target leaf. Continuous attributes are first discretised into 5 quantile bins, so all splits are performed on ordinal bins. The algorithm then deduces the hash fingerprint Φ corresponding to the next state for each action and queries the state transposition table $\mathcal{T}$ for an $\mathcal{O}(1)$ complexity check to achieve state deduplication. For retained valid actions, the algorithm records only a lightweight *action recipe* for each child node, mounting it to the search tree to achieve delayed memory allocation. Once all child nodes are simulated and created, the algorithm normalizes their corresponding Gini impurity decreases using a Softmax function to generate the prior probabilities P required for subsequent searches.

Simulate: Instead of employing traditional MCTS random rollouts, this paper directly uses the single-step split reward as the initial estimate for the backward split reward. Combining the Global Leaf Registry (GLR) mechanism from Section 3 with the "compute one side, deduce the other" logic, the algorithm estimates the single-step split accuracy reward r and Gini gain for the valid actions generated in the Expand phase based on the full training data.

Backup: Given the deterministic nature of the environment, the algorithm updates the expected value Q and visit count N of traversed nodes bottom-up along the search path using a Max-Backup strategy. This ensures that information from high-value branches propagates upwards rapidly, guiding subsequent iterations to focus on high-quality search regions.

4.2 PRIOR-GUIDED PUCT EXPLORATION

In the binary splitting of decision trees, when facing an action space comprised of massive feature and threshold combinations, classic Upper Confidence Bound (UCB) strategies tend to waste significant computational resources on low-potential split actions initially due to a lack of directional guidance. In modern reinforcement learning paradigms (e.g., AlphaZero), the output of a deep neural network is typically utilized as the prior probability $P(s, a)$ to guide tree search directions. Considering inference overhead and model interpretability, PG-DTS directly extracts single-step heuristic information to construct the prior guidance.

Specifically, for a candidate action $a = (l, X_i, th)$ under state s, the algorithm calculates the Gini impurity decrease $G(s, a)$ resulting from the split during the Simulate phase. It then applies a normalization mapping incorporated with a Laplace smoothing term ϵ to convert it into a prior

probability distribution $P(s, a)$:

$$P(s,a) = \frac{G(s,a) + \epsilon}{\sum_{a' \in \mathcal{A}(s)} (G(s,a') + \epsilon)}. \tag{2}$$

In the subsequent Select phase (Algorithm 2), the algorithm follows the PUCT rule to select actions:

$$a^* = \arg\max_{a \in \mathcal{A}(s)} \left(Q(s,a) + c \cdot P(s,a) \frac{\sqrt{N(s)}}{1 + N(s,a)} \right), \tag{3}$$

where $Q(s, a)$ is the cumulative expected value of the action, N represents the node visit count, and c is the exploration constant.

This mechanism combines greedy split information with global exploration. Early in the search, high-gain actions receive larger priors and are explored first. As those actions accumulate visits, the exploration term decays, allowing PG-DTS to revisit lower-prior branches that may have weak immediate gain but higher downstream value, as in feature-interaction patterns. Thus, the prior focuses the finite search budget without turning the procedure into a purely greedy algorithm.

4.3 Structure-aware space compression

In the context of decision tree construction, the scalability of search algorithms is hindered by the repetitive exploration of equivalent states and the instantiation memory overhead of massive search nodes. To address this, PG-DTS introduces a dual structure compression mechanism to optimize the utilization of computational resources, as detailed in Algorithm 3.

State Transposition Deduplication: In decision tree construction, differing feature split sequences can produce data distributions with entirely identical partitions of the feature space. Treating such structurally equivalent search nodes as independent states leads to massive redundant branches within the search space, squandering computational resources.

To counter this, we define the global hash fingerprint $\Phi(s)$ of a search node s as the unordered set hash of the signatures $\phi(l)$ of all leaf nodes contained within it:

$$\Phi(s) = \mathrm{Hash}\left(\{\phi(l) \mid l \in \mathcal{L}(s)\}\right). \tag{4}$$

This fingerprint uniquely characterizes a specific partition scheme of the feature space. During the Expand phase, the algorithm calculates the hash fingerprint $\Phi(s)$ corresponding to the expected state of a candidate split action. If this fingerprint already exists in the state transposition table $\mathcal{T}$, it indicates that this partitioning scheme has already been explored, and the algorithm will directly discard this redundant action, effectively compressing the search space.

Lazy-Build Mechanism: Facing hundreds or thousands of valid child nodes generated during each Expand phase, fully instantiating their corresponding internal decision tree objects in memory easily triggers Out-Of-Memory (OOM) errors. The algorithm decouples the creation of search nodes from the instantiation of their internal decision tree structures. During expansion, it only records lightweight "action recipes" (split features, thresholds, etc.) for valid child nodes. The algorithm instantiates the decision tree inside the search node according to the recorded action recipe *only* when that virtual node is selected by the PUCT strategy for downward exploration in a subsequent Select phase, or when it is chosen during the inference phase to extract the final model. This "build-on-demand" mechanism ensures that memory resources are not occupied by massive un-explored peripheral branches, thereby mitigating OOM risks caused by action space expansion.

The combination of state deduplication and lazy building reduces memory spent on redundant or unexplored branches, making the fixed-budget search more practical in high-dimensional action spaces.

5 Experiments

We evaluate PG-DTS along four questions aligned with the main claims: (1) whether it can escape local optima caused by feature interactions; (2) how it compares with greedy and exact tree learners on benchmark tabular datasets; (3) whether GLR, prior guidance, deduplication, and lazy instantiation improve search efficiency; and (4) whether the resulting trees retain compact and stable rule structures.

5.1 EXPERIMENTAL SETUP

Baselines & Metrics: We compare PG-DTS against CART, exact solvers (GOSDT, DL8.5), and the MDP-based DPDT. The primary metric is test set accuracy, evaluated across 4 synthetic scenarios (testing local optima escape and robustness) and 12 datasets sourced from the OpenML platform (Vanschoren et al., 2013). Detailed characteristics, OpenML identifiers, and original citations for these benchmark datasets are provided in Appendix E, and the synthetic generators are specified in Appendix D. Synthetic results are averaged over 30 runs, while benchmark data utilizes 5-fold cross-validation. All baselines were run with their default settings, except for the parameters needed to place every method in the same model class. CART and DPDT were trained to a maximum depth of d with a fixed random seed. DL8.5 was given the same depth limit and a 300 s time budget. GOSDT was run with a depth budget of d, a 300 s time limit, and a regularization of $1/n$, where n is the number of training samples in the fold, with its small-regularization option enabled.

Implementation Details: The search budget M is the maximum number of search iterations, i.e. the number of Select–Expand–Backup cycles the algorithm is allowed to perform; it is a compute budget rather than a stopping criterion based on tree quality. Because the number of candidate split actions grows with the tree depth, M is set jointly with the maximum depth: on the benchmark datasets we use $(d, M) = (2, 1000)$, $(3, 3000)$ and $(4, 5000)$, and the simulated scenarios use $M = 5000$. Each PG-DTS run is additionally capped by a wall-clock limit of 1200 s. While exact solvers require binary inputs—leading to feature space explosion through one-hot encoding—PG-DTS natively supports splitting on ordinal features, better preserving their natural semantics (Lin et al., 2020; Aglin et al., 2020). Experiments were run on a 25-core Intel Xeon machine with 90 GB of RAM.

5.2 ESCAPING LOCAL OPTIMA AND ROBUSTNESS

This section evaluates the algorithms' optimization capabilities against purely interactive features and their robustness against label noise, using simulated datasets ($d = 4$). Results for both regular and noisy scenarios are comprehensively presented in Table 1.

Escaping Local Optima: In the XOR trap scenario, which lacks explicit directional heuristic information, the greedy algorithm (CART) suffers from short-sightedness and halts partitioning prematurely (achieving only 53.72% accuracy). In contrast, exact global solvers (GOSDT, DL8.5) and the dynamic programming algorithm (DPDT) successfully capture the true interactive logic. Notably, relying on its prior guidance mechanism, PG-DTS also successfully escapes the local optimum, achieving 70.42% accuracy and demonstrating an optimization upper bound highly consistent with exact solvers.

Noise Robustness: When 20% label noise is injected, the exact solvers show weaker generalization on several synthetic tasks, suggesting that optimizing the finite training state space can become sensitive to noisy partitions under the fixed depth setting. PG-DTS obtains the highest test accuracy across the noisy scenarios in Table 1. A likely reason is that the PUCT prior gives low priority to splits with weak single-step evidence, while still allowing later exploration when downstream rewards justify it.

5.3 GENERALIZATION AND SCALABILITY ON BENCHMARK DATA

We next evaluate all methods with 5-fold cross-validation ($d = 4$) on 12 OpenML datasets with different sample sizes and feature dimensions. The results are summarized in Table 2.

Predictive performance: PG-DTS obtains the best accuracy on 7 of the 12 datasets, including `adult`, `breast-cancer`, and `credit-approval`, and is competitive on most of the remaining datasets. The results support the central claim that global search can improve over greedy split selection on a range of tabular problems, although the gains are dataset-dependent.

Scalability: The exact solvers encounter failures on some larger or higher-dimensional datasets under the experimental budget, marked as "—" in Table 2. This is consistent with the known cost of binarization and branch-and-bound search on expanded feature spaces. PG-DTS completes all 12 runs under the same depth setting, which supports its role as a practical approximate global search method rather than an exact replacement for optimal solvers.

Table 1: Test set accuracy (%) comparison on synthetic scenarios ($d = 4, N = 5000$).

Dataset	CART	DL8.5	DPDT	GOSDT	PG-DTS
Regular Scenarios ($\eta = 5\%$)					
DNF	**95.08 ± 0.24**	**95.08 ± 0.24**	**95.08 ± 0.24**	**95.08 ± 0.24**	**95.08 ± 0.24**
Tree_layer3	94.22 ± 0.49	**95.17 ± 0.23**	**95.17 ± 0.23**	**95.17 ± 0.23**	**95.17 ± 0.23**
Tree_layer4	89.05 ± 1.49	**95.14 ± 0.26**	92.65 ± 0.92	**95.14 ± 0.26**	93.70 ± 0.74
Voting	77.99 ± 0.62	78.60 ± 0.35	**78.99 ± 0.42**	78.61 ± 0.31	75.63 ± 0.50
XOR	53.72 ± 2.24	70.22 ± 0.45	65.42 ± 2.03	70.22 ± 0.45	**70.42 ± 0.47**
Label Noise Scenarios ($\eta = 20\%$)					
DNF	74.40 ± 2.59	69.07 ± 2.04	75.13 ± 2.35	70.93 ± 2.04	**76.50 ± 2.13**
Tree_layer3	75.47 ± 1.45	69.43 ± 1.92	75.77 ± 1.76	71.73 ± 2.00	**77.63 ± 1.62**
Tree_layer4	69.80 ± 2.01	71.57 ± 1.76	71.07 ± 2.23	72.53 ± 1.72	**73.90 ± 1.98**
Voting	62.03 ± 1.76	61.30 ± 2.03	63.40 ± 1.81	61.87 ± 2.09	**63.70 ± 1.73**
XOR	53.90 ± 2.02	60.03 ± 2.13	56.73 ± 2.37	60.13 ± 2.16	**61.53 ± 1.68**

Table 2: Mean test accuracy (%) comparison on the benchmark datasets using 5-fold cross-validation ($d = 4$).

Dataset	CART	DL8.5	DPDT	GOSDT	PG-DTS
adult	83.74	83.68	83.97	—	**83.99**
balance-scale	77.60	71.36	**79.52**	70.88	74.24
breast-cancer	94.85	93.85	94.28	94.42	**96.14**
credit-approval	83.33	82.75	81.74	83.04	**83.62**
dermatology	89.62	92.61	89.08	75.43	**93.17**
diabetes	73.96	73.30	73.70	70.71	**74.61**
heart-disease	76.91	77.23	72.93	**80.18**	79.86
ionosphere	88.31	86.58	86.59	**89.18**	88.89
sonar	71.15	70.26	70.69	—	**75.52**
spect	76.79	76.81	76.39	77.14	**80.89**
splice	**92.10**	89.40	**92.10**	—	92.07
congressional-voting	95.63	93.33	94.02	**96.09**	95.63

Note: "—" indicates the algorithm failed to return a result due to OOM or timeout.

5.4 Ablation studies

We evaluate the contributions of key components by comparing the full PG-DTS (Baseline) against variants without the Global Leaf Registry (w/o GLR), state deduplication (w/o Dedup), and prior guidance (w/o Prior).

Convergence efficiency: The *w/o Prior* variant shows a slower start, suggesting that heuristic priors help allocate the early search budget. The *w/o Dedup* variant reaches lower plateaus in deeper logic settings, consistent with redundant exploration of transposed states. The full PG-DTS combines both mechanisms and reaches higher root values within the same iteration budget.

Computational acceleration: Removing GLR increases end-to-end runtime by approximately 300% across the tested distributions. This indicates that cached leaf statistics and complement-based updates are important for making full-batch tree search practical.

5.5 Case study: interpretability in clinical diagnosis

As an interpretability case study, we examine decision rules learned on the Breast Cancer dataset ($d = 2$), where compact and stable rules are easier for domain experts to inspect. Table 3 shows one fold, with additional folds reported in Appendix C.

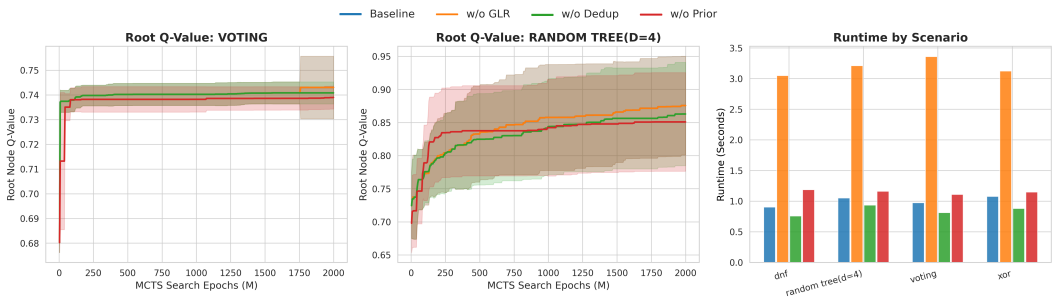


Figure 2: Ablation study on convergence efficiency and computational time. The left and middle charts illustrate the evolution of Root Q-Value over iterations, where the blue and green lines eventually overlap. Shaded bands in these two charts indicate ± 1 standard deviation over the 20 independent runs on the synthetic scenarios ($N = 5000$). The right chart displays the end-to-end execution time.

All methods identify *Bare_Nuclei* as an important feature, which is consistent with prior breast cytology knowledge (Wolberg & Mangasarian, 1990). CART shows more variation in root features and thresholds across folds. PG-DTS tends to produce interval-based rules with a more stable backbone, while GOSDT often yields exact one-hot conditions such as $== 10.0$ after binarization. This example illustrates how the search and representation choices affect the readability of the final tree, rather than serving as clinical validation.

Table 3: Decision rules extracted from the Breast Cancer dataset ($d = 2$).

Method	Root Node	Left Node (root=true)	Right Node (root=false)
PG-DTS	Bare_Nuclei $\in \{1.0, 2.0\}$	Cell_Size_Uniformity $\in \{1.0, 2.0, 3.0, 4.0\}$ (yes $\rightarrow$ class0, no $\rightarrow$ class1)	Cell_Shape_Uniformity $\in \{1.0, 2.0\}$ (yes $\rightarrow$ class0, no $\rightarrow$ class1)
CART	Cell_Shape_Uniformity ≤ 3.500	Bare_Nuclei ≤ 5.500 (yes $\rightarrow$ class0, no $\rightarrow$ class1)	Bare_Nuclei ≤ 1.500 (yes $\rightarrow$ class1, no $\rightarrow$ class1)
GOSDT	Bare_Nuclei $== 1.0$	$\rightarrow$ class0	Cell_Size_Uniformity $== 10.0$ (yes $\rightarrow$ class0, no $\rightarrow$ class1)

6 Conclusion

This paper proposes PG-DTS, a Prior-Guided Global Decision Tree Search framework for inducing interpretable axis-aligned trees beyond greedy split selection. By formulating tree induction as a deterministic MDP and combining PUCT prior guidance with leaf-level reward reuse, state deduplication, and lazy instantiation, PG-DTS searches over tree structures under a practical computational budget. Empirical results show that PG-DTS escapes synthetic interaction traps that challenge CART, remains competitive on benchmark tabular datasets, and avoids several runtime or memory failures encountered by exact solvers. Overall, the results suggest that prior-guided global search can make decision trees less myopic while keeping their rules explicit and auditable.

AI use statement

In this work, we used generative AI tools for one task with required disclosure under the ICLR 2027 AI Policy for Authors, namely assisting with the translation of manuscript text. We have not used generative AI tools for any other task with required disclosure, and the remaining required disclosure tasks are not applicable to this work. Specifically, no generative AI was used to generate synthetic data sets, to develop the theoretical models or conceptual frameworks, to formulate mathematical claims, to provide critical ingredients for or assist in the writing of proofs, to propose or refine hypotheses, to design or provide feedback on the research methodology or experiments, to implement

the proposed method, to clean or reformat datasets, to support qualitative or thematic data analysis, or to interpret the results.

Additionally, we used generative AI tools for tasks with recommended disclosure, namely drafting parts of the manuscript and editing it to improve readability. The AI-drafted parts are limited to expository text that presents work the authors had already carried out. They were reviewed and, where necessary, corrected by the authors.

We have reviewed all AI-assisted work. The research idea, the MDP formulation, the algorithm design, the implementation, the experiments, and the theoretical analysis were conceived and carried out by the authors. AI-assisted text was checked against our implementation and the cited literature, and all references were verified against the original sources. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

Ethics statement

In high-stakes domains such as clinical decision support and finance, intrinsically interpretable models can make auditing easier than black-box alternatives. PG-DTS contributes to this direction by improving the search procedure for axis-aligned trees, which may help practitioners obtain more accurate models without giving up explicit, auditable decision rules. The decision-rule case study in Section 5.5 uses the publicly available Breast Cancer dataset and is intended only to illustrate the readability of the learned rules; it is not clinical validation and should not be interpreted as medical evidence.

At the same time, interpretability alone does not guarantee fairness, safety, or clinical utility. Labels in datasets such as the ones used here may encode historical bias, and an interpretable model can still reproduce and make such bias easier to act on. Deployment in any regulated setting would therefore require task-specific validation, monitoring, and bias assessment beyond what is reported in this paper. We release no new datasets or models, and we use only public benchmark data with appropriate citations. There are no conflicts of interest and no sponsorship to declare.

Reproducibility statement

We have made the following efforts to ensure that the results reported in this paper can be reproduced. The full problem formulation, including the deterministic MDP and the locality-based reward decomposition, is given in Section 3. Pseudocode for the four search phases—Select, Expand, Simulate, and Backup—along with the DeriveStats subroutine, is provided in Appendix A, and the link between the maximized value function and global accuracy is proved in Appendix B. The 12 benchmark OpenML datasets, including their OpenML identifiers, feature counts, and sample counts, are listed in Appendix E, together with the full set of extracted decision rules used for the case study in Appendix C. Anonymized source code implementing PG-DTS and the baselines are available at `https://anonymous.4open.science/r/pgdts-4828/`. All baselines are publicly available and were run with their default settings under the same depth constraint.

References

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

Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*, 58:82–115, 2020.

Rodrigo Coelho Barros, Márcio Porto Basgalupp, Andre CPLF De Carvalho, and Alex A Freitas. A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(3):291–312, 2011.

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

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

Miguel Á Carreira-Perpiñán and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pp. 1211–1221, 2018.

Ayman Chaouki, Jesse Read, and Albert Bifet. Online learning of decision trees with thompson sampling. In *International Conference on Artificial Intelligence and Statistics*, pp. 2944–2952. PMLR, 2024.

Benjamin E Childs, James H Brodeur, and Levente Kocsis. Transpositions and move groups in Monte Carlo tree search. In *2008 IEEE Symposium On Computational Intelligence and Games*, pp. 389–395. IEEE, 2008.

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

Dheeru Dua and Casey Graff. UCI machine learning repository, 2019. URL `http://archive.ics.uci.edu/ml`.

Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI magazine*, 38(3):50–57, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Hector Kohler, Riad Akrour, and Philippe Preux. Breiman meets bellman: Non-greedy decision trees with mdps. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '25)*. ACM, 2025.

Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. Generalized and scalable optimal sparse decision trees. In *International conference on machine learning*, pp. 6150–6160. PMLR, 2020.

Marvin Minsky and Seymour Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 1969.

Cecília Nunes, Mathieu De Craene, Hélène Langet, Oscar Camara, and Anders Jonsson. Learning decision trees through monte carlo tree search: An empirical evaluation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(3):e1348, 2020.

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

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

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

Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2013.

William H Wolberg and Olvi L Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the national academy of sciences*, 87(23): 9193–9196, 1990.

A DETAILED ALGORITHMS

Algorithm 2 Selection phase (SELECT)

Input: Root node s_{root}
Output: Boundary node s
$s \leftarrow s_{\text{root}}$
while s.children $\neq \emptyset$ **do**
  $s \leftarrow \text{argmax}_{s' \in s.\text{children}} \left(s'.Q + C_{\text{puct}} \cdot s'.P \frac{\sqrt{s.N}}{1+s'.N} \right)$ ▷ PUCT selection
  if s.dt_state == NULL **then**
    $(l, X_i, th) \leftarrow s$.split_recipe ▷ Lazy build, instantiate on-demand
    s.dt_state $\leftarrow$ PHYSICALSPLIT(s.parent.dt_state, l, X_i, th)
  end if
end while
return s

Algorithm 3 Expansion phase (EXPAND)

Input: Current node s
s.is_expand $\leftarrow$ True
$children_list \leftarrow \emptyset$
for each $leaf \in s$.dt_state.leaves **do**
  if $leaf$.layer $\geq$ max_layer **or** $leaf.h \geq 0.999$ **or** $leaf.p \leq 0.001$ **then**
    continue ▷ Filter leaves exceeding max depth, meeting purity, or with low sample ratio
  end if
  ▷ Get valid splits: filter dead features and match candidate thresholds
  ▷ Candidate thresholds of a feature = its unique discretised values except the largest, restricted to the leaf's observed range
  ▷ Continuous attributes are first discretised into 5 quantile bins; every split is $X \leq th$
  $\mathcal{C}_{\text{actions}} \leftarrow$ GENERATECANDIDATES($leaf$)
  for each $(f, th) \in \mathcal{C}_{\text{actions}}$ **do**
    $state_hash \leftarrow s$.SIMULATESPLIT($leaf, f, th$)
    if $\mathcal{T}$.ISDUPLICATE($state_hash$) **then**
      continue ▷ State deduplication
    end if
    $X_{\text{curr}} \leftarrow X[leaf$.indices, $f]$
    $r, gain \leftarrow$ SIMULATE($leaf, f, th, X_{\text{curr}}$) ▷ Simulate to compute rewards
    $child \leftarrow s$.ADDLAZYCHILD($leaf, f, th, r, gain$) ▷ Mount lazy child node
    $children_list$.APPEND($child$)
  end for
end for
if $children_list == \emptyset$ **then**
  s.is_terminal $\leftarrow$ True
  return
end if
$Priors \leftarrow$ SOFTMAX($[c.gain$ for $c \in children_list], \tau$) ▷ Compute prior probabilities
for each $(i, c) \in$ enumerate($children_list$) **do**
  $c.P \leftarrow Priors[i]$
end for

Algorithm 4 Simulation phase (SIMULATE)

Input: Parent leaf node l, split feature f, threshold th, feature value data X_{curr}
Output: Accuracy reward r, Gini impurity decrease $gain$
$l_{\text{left}} \leftarrow l \cup \{(f, '\leq', th)\}$
$l_{\text{right}} \leftarrow l \cup \{(f, '>', th)\}$
if $l_{\text{left}} \in \mathcal{L}$ **and** $l_{\text{right}} \notin \mathcal{L}$ **then**
    REGISTERLEAF($\mathcal{L}, l_{\text{right}}$)
    $l_{\text{right}} \leftarrow$ DERIVESTATS($l, l_{\text{left}}, l_{\text{right}}$) ▷ Deduce right node via left node
else if $l_{\text{left}} \notin \mathcal{L}$ **and** $l_{\text{right}} \in \mathcal{L}$ **then**
    REGISTERLEAF($\mathcal{L}, l_{\text{left}}$)
    $l_{\text{left}} \leftarrow$ DERIVESTATS($l, l_{\text{right}}, l_{\text{left}}$) ▷ Deduce left node via right node
else if $l_{\text{left}} \notin \mathcal{L}$ **and** $l_{\text{right}} \notin \mathcal{L}$ **then**
    REGISTERLEAF($\mathcal{L}, l_{\text{left}}$)
    REGISTERLEAF($\mathcal{L}, l_{\text{right}}$)
    ▷ Calculate left node statistics based on parent data
    $mask \leftarrow (X_{\text{curr}} \leq th)$
    l_{left}.indices $\leftarrow l$.indices$[mask]$
    l_{left}.classcount $\leftarrow$ BINCOUNT($Y[l_{\text{left}}$.indices$]$)
    l_{left}.size $\leftarrow$ sum(l_{left}.classcount)
    $l_{\text{left}}.p \leftarrow l.p \times (l_{\text{left}}$.size$/l$.size$)$
    $l_{\text{left}}.h \leftarrow \max(l_{\text{left}}$.classcount$)/l_{\text{left}}$.size
    l_{left}.gini $\leftarrow 1.0 -$ sum(l_{left}.classcount^2)$/(l_{\text{left}}$.size^2)
    ▷ Deduce right node via left node
    $l_{\text{right}} \leftarrow$ DERIVESTATS($l, l_{\text{left}}, l_{\text{right}}$)
end if
$r \leftarrow (l_{\text{left}}.p \times l_{\text{left}}.h + l_{\text{right}}.p \times l_{\text{right}}.h) - (l.p \times l.h)$
$gain \leftarrow l$.gini $- (l_{\text{left}}$.gini $\times\, l_{\text{left}}.p + l_{\text{right}}$.gini $\times\, l_{\text{right}}.p)/l.p$
return $r, gain$

Algorithm 5 Node statistics derivation (DERIVESTATS)

Input: Parent leaf l, source child l_{src} with known stats, target child l_{tgt} to derive
Output: Target node l_{tgt} with completed statistics
l_{tgt}.indices $\leftarrow l$.indices $\setminus\, l_{\text{src}}$.indices
l_{tgt}.classcount $\leftarrow l$.classcount $- l_{\text{src}}$.classcount
l_{tgt}.size $\leftarrow$ sum(l_{tgt}.classcount)
$l_{\text{tgt}}.p \leftarrow l.p - l_{\text{src}}.p$
$l_{\text{tgt}}.h \leftarrow \max(l_{\text{tgt}}$.classcount$)/l_{\text{tgt}}$.size
l_{tgt}.gini $\leftarrow 1.0 -$ sum(l_{tgt}.classcount^2)$/(l_{\text{tgt}}$.size^2)
return l_{tgt}

Algorithm 6 Backup phase (BACKUP)

Input: Expanded boundary node s
while $s \neq$ NULL **do**
  $s.N \leftarrow s.N + 1$ ▷ Increment path visit counts
  if s.children $\neq \emptyset$ **then**
    $s.Q \leftarrow s.R + \max_{s' \in s.\text{children}} s'.Q$ ▷ Deterministic max value upward propagation
  end if
  $s \leftarrow s$.parent
end while

B OPTIMALITY OF VALUE MAXIMIZATION

In this section, following the theoretical groundwork established in Chaouki et al. (2024), we show that maximizing the value function in the formulated MDP is equivalent to maximizing the chosen tree-accuracy objective over the reachable tree space. This result characterizes the objective optimized by the search; the finite-budget PG-DTS algorithm is an approximate search procedure for this objective.

Theorem 1. *The optimal policy π^* of the proposed MDP generates a decision tree T^* that maximizes the global reward function $H_r(\cdot)$.*

Proof. Let R denote the initial root state (s_{root}). Since the decision tree induction process is bounded by the maximum depth and the finite number of feature splits, the corresponding MDP has finite state and action spaces. Consequently, an optimal policy exists and can be chosen to be deterministic:

$$\pi^* \in \operatorname*{argmax}_{\pi} V^{\pi}(R). \tag{5}$$

Let the trajectory generated by the optimal policy π^* be denoted as:

$$R \xrightarrow{\pi^*} s_1 \xrightarrow{\pi^*} \cdots \xrightarrow{\pi^*} T^*, \tag{6}$$

where T^* represents the final tree structure before termination. Recall that our cumulative reward function is telescoping (Section 3.2). The value of the root state under policy π^* is equivalent to the accuracy of the resulting tree:

$$H(T^*) - H(R) = V^{\pi^*}(R) \geq V^{\pi}(R), \quad \forall \pi. \tag{7}$$

Now, consider any arbitrary valid decision tree T reachable from the root R. There exists a policy π (a sequence of split actions) such that:

$$R \xrightarrow{\pi} s'_1 \xrightarrow{\pi} \cdots \xrightarrow{\pi} T. \tag{8}$$

By the same logic of the telescoping reward sum, the value of the policy generating T is:

$$V^{\pi}(R) = H(T) - H(R). \tag{9}$$

Combining Eq. equation 7 with the existence of such a policy for any T, we obtain:

$$H(T^*) \geq H(T). \tag{10}$$

Thus, T^* is the global optimal decision tree. □

C DETAILED DECISION RULES FOR BREAST CANCER DATASET (5-FOLD)

Table 4 presents the full decision rules extracted from the Breast Cancer dataset across all five cross-validation folds.

D SYNTHETIC DATA GENERATION

This section specifies how the synthetic scenarios evaluated in Section 5.2 are generated, so that the study can be reproduced exactly. All scenarios share a single generation pipeline and differ only in the rule that maps features to labels.

Table 4: Decision rules comparison across 5-fold cross-validation ($d = 2$).

Method	Fold	Root Node	Left Node (root=true)	Right Node (root=false)
PG-DTS	1	Bare_Nuclei $\in \{1, 2\}$	Cell_Size_Uniformity $\in \{1, 2, 3, 4\}$	Cell_Shape_Uniformity $\in \{1, 2\}$
	2	Cell_Size_Uniformity $\in \{1\}$	Bare_Nuclei $\in \{1, 2\}$	Bare_Nuclei $\in \{1, 2\}$
	3	Bare_Nuclei $\in \{1, 2\}$	Cell_Size_Uniformity $\in \{1, 2, 3\}$	Cell_Shape_Uniformity $\in \{1, 2\}$
	4	Bare_Nuclei $\in \{1, 2\}$	Cell_Size_Uniformity $\in \{1, 2, 3\}$	Cell_Shape_Uniformity $\in \{1\}$
	5	Bare_Nuclei $\in \{1, 2\}$	Cell_Size_Uniformity $\in \{1, 2, 3\}$	Cell_Shape_Uniformity $\in \{1, 2\}$
CART	1	Cell_Shape_Uniformity ≤ 3.5	Bare_Nuclei ≤ 5.5	Bare_Nuclei ≤ 1.5
	2	Bare_Nuclei ≤ 2.5	Cell_Size_Uniformity ≤ 3.5	Cell_Size_Uniformity ≤ 1.5
	3	Cell_Size_Uniformity ≤ 3.5	Bare_Nuclei ≤ 5.5	Bare_Nuclei > -4999.0
	4	Cell_Size_Uniformity ≤ 2.5	Bare_Nuclei ≤ 4.5	Cell_Shape_Uniformity ≤ 2.5
	5	Cell_Size_Uniformity ≤ 2.5	Normal_Nucleoli ≤ 3.5	Cell_Shape_Uniformity ≤ 2.5
GOSDT	1	Bare_Nuclei $== 1.0$	$\rightarrow$ Class 0	Cell_Size_Uniformity $== 10.0$
	2	Bare_Nuclei $== 1.0$	$\rightarrow$ Class 0	Bland_Chromatin $== 8.0$
	3	Bare_Nuclei $== 1.0$	$\rightarrow$ Class 0	Cell_Size_Uniformity $== 10.0$
	4	Bare_Nuclei $== 1.0$	$\rightarrow$ Class 0	Cell_Size_Uniformity $== 10.0$
	5	Single_Epi_Cell_Size $== 2.0$	$\rightarrow$ Class 0	Bare_Nuclei $== 10.0$

Pipeline. Every scenario is a generator function parameterised by the size of the sample pool N, the number of features p, the number of values v_j allowed for each feature j, the number of classes k, a label-noise level η, and a flag controlling feature shuffling. Generation proceeds in three stages.

- *Feature sampling.* The feature matrix X is drawn first, independently per column: entry X_{ij} is sampled uniformly from $\{0, \ldots, v_j - 1\}$. The experiments in Section 5.2 use $p = 20$ and $v_j = 2$ for every j, so every feature is binary; the number of classes is $k = 2$ and the pool size is $N = 5000$. The ablation study in Section 5.4 uses $p = 40$.
- *Label assignment.* The clean label Y is computed from X by the scenario rule given below.
- *Label noise.* Each label is independently replaced with probability η by $(Y + \delta) \bmod k$, where δ is drawn uniformly from $\{1, \ldots, k - 1\}$; a noisy label therefore never coincides with the clean label. The regular scenarios use $\eta = 5\%$, and the noise-robustness scenarios use $\eta = 20\%$.

After the labels have been computed, the columns of X are randomly permuted, so that the features carrying the rule are not systematically the first ones and the search cannot exploit feature position.

Train/test split. The pool of $N = 5000$ instances is split once, uniformly at random, into 80% training and 20% test data (4000/1000) using the seed of that run, and the same split is shared by all methods within the run. Each scenario is repeated over 30 independent runs with different seeds $(0, \ldots, 29)$; a fresh pool and a fresh split are drawn for each run, and we report the mean test accuracy over the 30 runs.

Scenario rules. Denote by $\mathrm{mid}_j = (v_j - 1)/2$ the median level of feature j. The scenarios reported in Table 1 are defined as follows.

- **DNF** is a disjunction of two conjunctions over four distinct features. It assigns $Y = 1$ if either $(X_0 \leq \mathrm{mid}_0 \wedge X_1 \leq \mathrm{mid}_1)$ or $(X_2 > \mathrm{mid}_2 \wedge X_3 > \mathrm{mid}_3)$ holds, and $Y = 0$ otherwise; this rule is stated for $k = 2$, the only setting used in our experiments. Each term alone covers roughly a quarter of the instance space, so the positive region is the union of two axis-aligned boxes; a greedy learner typically commits to one term and stops.
- **XOR** is the interaction trap. For features 0 and 1, the conditions cond_0 and cond_1 are set to *true* above the respective median and *false* below it, while instances whose value falls exactly on the median are assigned by a fair coin flip. This keeps each condition at $50/50$ for any v_j. The label is $Z = \mathrm{cond}_0 \oplus \mathrm{cond}_1$ for $k = 2$ (for $k > 2$, $Z = 0$ is mapped uniformly to the even classes and $Z = 1$ to the odd classes, which preserves the symmetry). Since neither condition alone carries any information about the label, every single-feature split has near-zero marginal information gain, which is exactly the configuration in which greedy induction stops prematurely.
- **Tree_layer3** and **Tree_layer4** are random axis-aligned trees of depth 3 and 4, labelled "random tree(d=3)" and "random tree(d=4)" in the figures. A tree is generated recursively:

each internal node selects a feature uniformly at random and a threshold uniformly from $\{1, \ldots, v_j - 1\}$, instances with $X_j < \texttt{threshold}$ descend to the left child and the remaining ones to the right, and each leaf is assigned a class uniformly at random from the k classes. Labels are the predictions of this tree, so an optimal axis-aligned tree of the given depth exists by construction and the task is exactly representable.

- **Voting** is a rank-sum rule. The levels of the first five features are summed and divided by the largest attainable sum $\sum_{j<5}(v_j - 1)$, yielding a score in $[0, 1]$ that is cut into k equal-width intervals to produce the label. This is a graded, additive rule that requires coordinating several features rather than relying on a single dominant split.

E BENCHMARK DATASETS DETAILS

Table 5 summarizes the characteristics of the 12 benchmark datasets utilized in our experiments. All datasets are publicly accessible via the OpenML platform (Vanschoren et al., 2013). We acknowledge the original creators and donors of these datasets, most of which originate from the UCI Machine Learning Repository (Dua & Graff, 2019). Dataset names follow their OpenML identifiers: the entry *congressional-voting* is the Congressional Voting Records dataset, which is unrelated to the synthetic *Voting* scenario defined in Appendix D.

Table 5: Details of the benchmark datasets sourced from OpenML.

Dataset	**OpenML ID**	**Features**	**Samples**
adult	1590	14	48,842
balance-scale	11	4	625
breast-cancer	15	9	699
credit-approval	29	15	690
dermatology	35	34	366
diabetes	37	8	768
heart-disease	45	13	303
ionosphere	59	34	351
sonar	40	60	208
spect	334	22	267
splice	46	60	3,190
congressional-voting	56	16	435