Approximate Optimal Active Learning of Decision Trees

Zunchen Huang and Chenglu Jin

Centrum Wiskunde & Informatica, Science Park 123, Amsterdam, The Netherlands
{zunchen.huang, chenglu.jin}@cwi.nl

Abstract. We consider the problem of actively learning an unknown binary decision tree using only membership queries, a setting in which the learner must reason about a large hypothesis space while maintaining formal guarantees. Rather than enumerating candidate trees or relying on heuristic impurity or entropy measures, we encode the entire space of bounded-depth decision trees symbolically in SAT formulas. We propose a symbolic method for active learning of decision trees, in which approximate model counting is used to estimate the reduction of the hypothesis space caused by each potential query, enabling near-optimal query selection without full model enumeration. The resulting learner incrementally strengthens a CNF representation based on observed query outcomes, and approximate model counter ApproxMC is invoked to quantify the remaining version space in a sound and scalable manner. Additionally, when ApproxMC stagnates, a functional equivalence check is performed to verify that all remaining hypotheses are functionally identical. Experiments on decision trees show that the method reliably converges to the correct model using only a handful of queries, while retaining a rigorous SAT-based foundation suitable for formal analysis and verification.

Keywords: Decision Tree · Active Learning · Approximate Model Counting

1 Introduction

Decision trees are a widely used model for representing Boolean functions across multiple domains, including program analysis [19, 21, 23, 44], verification [24, 28], and machine learning [13, 26, 37]. In formal methods scenarios, decision trees naturally arise when modeling temporal logic, classification decisions, or conditional behaviors in programs [1, 2, 10, 11, 14, 31]. Unlike deep neural network with complex, layered architectures and difficulty in interpreting how predictions are made, decision trees are interpretable, compact, and can represent arbitrary Boolean functions of bounded complexity, making them suitable for symbolic reasoning on various systems.

In many practical applications, we are confronted with a fundamental problem: given black-box access to a Boolean function implemented as a decision tree, can we efficiently reconstruct the underlying function with as few queries

as possible? Black-box access is typically achieved through a *membership oracle*, which evaluates the function on input provided by the learner. This setup is relevant in software testing [12,32,45], formal verification [3,34,38,39], and security analyses [9, 30, 33, 35], where direct inspection of the implementation may be impossible, expensive, unsafe, or breaking privacy commitment. However, controlled and highly selective queries can be made to infer systems behavior and useful properties efficiently.

Traditional approaches to active learning of Boolean functions fall into two main categories. First, *enumeration-based methods* attempt to explicitly maintain and prune the entire hypothesis space of candidate functions consistent with the observed queries [20,40]. While these methods are conceptually straightforward, they scale poorly: the number of candidate decision trees grows exponentially with the number of features and the tree depth, quickly becoming intractable even for moderate problem sizes. Second, *heuristic-based methods* select queries based on information gain, or other statistical criteria [8]. Although these heuristics can guide query selection without explicit enumeration, they provide no formal guarantees on the reduction of the hypothesis space and may require many redundant queries, especially when the underlying function exhibits intricate dependencies between features.

In this work, we introduce a symbolic, model counting guided approach that bridges the gap between formal methods and active learning, which is illustrated in Figure 1. The symbolic reasoning engine is within the blue dotted box. Our key insight is that the space of all decision trees of a given depth and feature set can be encoded compactly as a *propositional formula* in conjunctive normal form (CNF) from a user provided property specification, i.e., decision tree topology. Each model of the CNF corresponds to a valid decision tree, satisfying constraints on internal node feature selections and leaf output assignments. Membership queries are then used to prune this hypothesis space: every observed input-output pair imposes additional CNF constraints, eliminating all candidate trees inconsistent with the observation.

To efficiently estimate the remaining hypothesis space after each query, we leverage *approximate model counting*, using the state-of-the-art `ApproxMC` [17] algorithm. `ApproxMC` provides a scalable method for estimating the number of satisfying assignments of a CNF formula without exhaustively enumerating all models. By using the counts to estimate the effect of potential queries on the version space, our learner selects the input that approximately halves the number of consistent hypotheses. It is possible that `ApproxMC` is unable to reduce the hypothesis space at the final iteration rounds. The functional equivalence check is designed to verify whether two remaining hypotheses, e.g., two decision trees or logical models, are functionally identical. This is achieved by encoding that the outputs of the two models should differ at least on one input. A SAT solver is invoked to determine if this assumption is unsatisfiable. This module is crucial to ensuring the correctness of the learning process, where we can confidently collapse equivalent hypotheses into one and continue refining and terminate the learning process.

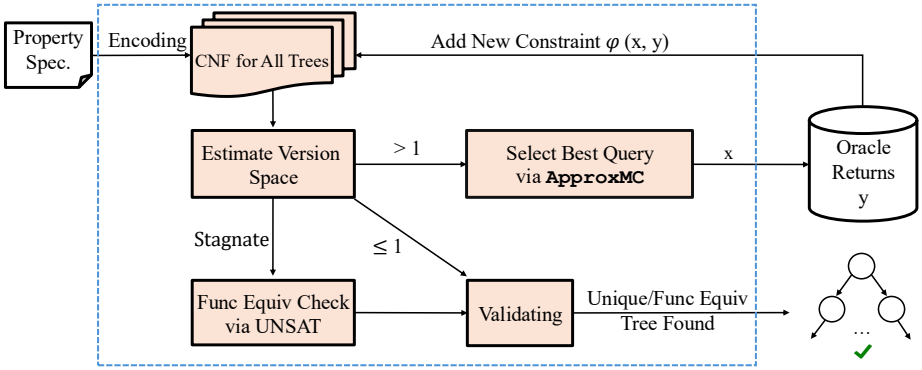

Fig. 1. SAMCAL: Approx. Model Counting Guided Symbolic Active Learning Method

By combining symbolic SAT-based reasoning with approximate model counting, our approach provides a rigorous framework for active learning in formal methods settings. Unlike purely heuristic methods, our technique maintains a formal representation of the version space, enabling guarantees about the reduction of hypotheses after each query. At the same time, approximate counting ensures scalability, avoiding the need for full enumeration of candidate decision trees. Consequently, this method represents a promising approach for tasks such as automated program analysis, testing of black-box components, and verification of systems with hidden components, where query efficiency and formal guarantees are both critical.

The main contributions of this work are as follows:

- **Formal encoding of decision tree hypothesis spaces.** We develop a systematic method for encoding all depth-d decision trees over n binary features. The resulting CNF is suitable for symbolic reasoning and approximate model counting.
- **Symbolic active learning framework.** We demonstrate how approximate model counting can guide query selection. Each candidate query is evaluated for its expected reduction in the hypothesis space, and the query maximizing this reduction is selected. This allows us to actively prune large hypothesis space efficiently.
- **Theoretical and practical analysis.** We show that each selected query approximately halves the hypothesis space, both theoretically and empirically. This near-halving property provides formal insight into why the method scales better than naive enumeration.
- **Evaluation on unknown decision trees.** We implement a Python tool, and it successfully reconstructs the hidden function, where the largest initial hypothesis space is over 10^{15}, using only a small number of membership queries, illustrating the feasibility and showing the scalability of our method.

The remainder of the paper is outlined as follows. In Section 2, we formalize the problem and analyze the version space with approximate model counting. In Section 3, we present the algorithm for approximate optimal active learning. In Section 4, we define the general SAT encoding . In Sections 5 and 6, we show the result on a motivation example and more general settings on the trees. We discuss the scalability, limitations, and related works in Sections 7 and 8. We finally present the conclusion in Section 9.

2 Problem Formulation

We formalize the active learning problem for decision trees. Let n denote the number of Boolean features, and d the depth of the decision tree.

2.1 Hypothesis Space

Let H denote the set of all decision trees of depth d over n Boolean features. Each hypothesis $h \in H$ is fully defined by:

- **Internal node assignments:** For each internal node i, a feature $T_i \in \{1, \ldots, n\}$ is selected.
- **Leaf outputs:** Each leaf ℓ is assigned a Boolean output $B_\ell \in \{0, 1\}$.

2.2 Version Space

Given a set of queries and observed oracle responses up to step t, the *version space* V_t is the subset of hypotheses consistent with all observations:

$$V_t = \{h \in H \mid h(x) = y, \forall (x, y) \text{ queried up to step } t\}.$$

2.3 Membership Oracle

We assume access to a black-box *membership oracle*:

$$\text{oracle} : \{0,1\}^n \to \{0,1\}, \quad x \mapsto f(x),$$

which returns the output of the unknown target function for any input x. The objective of active learning is to select a sequence of queries that efficiently shrinks V_t until $|V_t| = 1$, i.e., the target hypothesis is uniquely identified.

2.4 Approximate Halving Analysis

For any (remaining) candidate input x, let

$$V_t^0 = \{h \in V_t \mid h(x) = 0\}, \quad V_t^1 = \{h \in V_t \mid h(x) = 1\}.$$

The ideal query maximizes the minimum reduction in the version space:

$$x^* = \arg \max_{x \in \{0,1\}^n} \min(|V_t^0|, |V_t^1|).$$

2.5 Approximate Counting with `ApproxMC`

Exact computation of $|V_t^0|$ and $|V_t^1|$ is generally infeasible. We leverage *approximate model counting* (`ApproxMC`) to estimate these values:

$$\tilde{c}_0 = \text{ApproxMC}(V_t \wedge (h(x) = 0)), \quad \tilde{c}_1 = \text{ApproxMC}(V_t \wedge (h(x) = 1)).$$

We then select

$$x^* = \arg\max_x \min(\tilde{c}_0, \tilde{c}_1),$$

which approximately halves the version space at each step.

2.6 Geometric Decay of Version Space

Let V_{t+1} denote the version space after observing the oracle output for x^*. Then

$$|V_{t+1}| \approx \frac{1}{2}|V_t|,$$

assuming the query is near-optimal. Iteratively applying this strategy yields logarithmic query complexity in the size of the initial hypothesis space $|\mathcal{H}|$:

$$\text{Number of queries} \lesssim \log_2 |\mathcal{H}|.$$

This forms the theoretical foundation for our approximate optimal active learning framework for decision trees.

3 Approximate Optimal Active Learning Algorithm

We now present the algorithm for approximately optimal active learning of decision trees using model counting guided queries. The algorithm iteratively selects inputs that approximately halve the remaining version space, based on estimates from `ApproxMC`.

3.1 Algorithm Overview

We show the high-level procedure of our method in Algorithm 1. Let $\mathcal{H}$ be the hypothesis space of all decision trees of a given depth and feature set, and $V_t \subseteq \mathcal{H}$ the version space after t queries. At each step, the algorithm performs the following operations: For each unqueried input $x \in \{0,1\}^n$, use `ApproxMC` to estimate the number of hypotheses consistent with V_t that would output 0 or 1. The score of input x is defined as $\min(\tilde{c}_0, \tilde{c}_1)$. Choose the input x^* with the highest score, which approximately halves the version space. Obtain the output $y = \text{oracle}(x^*)$. Restrict V_t to hypotheses consistent with the new observation: $V_{t+1} = \{h \in V_t \mid h(x^*) = y\}$. Continue until the version space is reduced to a single hypothesis, i.e., $|V_t| = 1$, or a stagnation is found and all remaining trees are checked whether they are functional equivalent.

Algorithm 1 High Level Procedure of the Proposed Method SAMCAL

Input: Tree spec, ApproxMC, oracle
$\phi \leftarrow$ EncodeCNFTrees()
$V_t \leftarrow \{\phi\}$ {— Initial version space —}
while $|V_t| > 1$ **do**
  $x^* \leftarrow \text{ApproxMC}(V_t)$ {— Select optimal query —}
  $y^* \leftarrow \text{oracle}(x^*)$
  Add constraint ϕ_{x^*,y^*} to V_t
  $V_{t+1} \leftarrow \{h \in V_t \mid h(x^*) = y^*\}$
end while
Output: Tree $h^* \in V_t$

Algorithm 2 Approximate Optimal Active Learning

$\mathcal{H}$ = calculate_H(TreeSpec)
Initialize version-space CNF: $V \leftarrow \mathcal{H}$ (initial tree hypothesis space)
Initialize queried set: Queried $\leftarrow \emptyset$
$\tilde{C}_{\text{prev}} \leftarrow \text{ApproxMC}(V)$
round $\leftarrow 0$
while true **do**
  for all $x \in \{0,1\}^n \setminus$ Queried **do**
    $\tilde{c}_0 \leftarrow \text{ApproxMC}(V \wedge (h(x) = 0))$
    $\tilde{c}_1 \leftarrow \text{ApproxMC}(V \wedge (h(x) = 1))$
    $score(x) \leftarrow \min(\tilde{c}_0, \tilde{c}_1)$
  end for
  $x^* \leftarrow \arg\max_x score(x)$
  $y^* \leftarrow \text{oracle}(x^*)$
  Queried $\leftarrow$ Queried $\cup \{x^*\}$
  $V \leftarrow V \wedge (V_{x^*} = y^*)$
  $\tilde{C}_{\text{new}} \leftarrow \text{ApproxMC}(V)$
  {— Unique Hypothesis Found —}
  if $\tilde{C}_{\text{new}} \leq 1$ **then**
    return V
  end if
  {— Check for ApproxMC stagnation —}
  if $|\tilde{C}_{\text{new}} - \tilde{C}_{\text{prev}}| < 1$ **then**
    Construct two copies $V^{(1)}, V^{(2)}$ of V
    Let $G \leftarrow V^{(1)} \wedge V^{(2)} \wedge \exists x\ (f_1(x) \neq f_2(x))$
    if Solve(G) = UNSAT **then**
      return Any model of V (all remaining trees are functionally identical)
    end if
  end if
  $\tilde{C}_{\text{prev}} \leftarrow \tilde{C}_{\text{new}}$
  round $\leftarrow$ round + 1
  if round $\geq$ MAX_ROUND **then**
    return "NO UNIQUE TREE FOUND"
  end if
end while

3.2 Algorithm Pseudocode

In Algorithm 2, we show our algorithm in detail. We first calculate $\mathcal{H}$ from the tree specification, i.e., depth and feature numbers of tree topology. To make it general, we assume the tree is a full binary tree. In lines 2-3, we encode the unknown tree into a CNF formula and initialize a set Queried to store all the inputs that have been asked to the oracle. In line 4, we invoke the model counter to approximate the models for the first time. Then initialize the round variable. In the while-loop from line 6, we select the input that can approximately obtain the highest score and store it to x^* in line 12, which can maximize the min reduction in the remaining version space. Then we query the oracle and receive y^* in line 13. We add the queried input into the Queried set in line 14. And add the corresponding constraint $\phi(x^*, y^*)$ into the CNF formula and updated V in line 15. In line 16, we invoke the model counter again to approximate the size of the updated V. If $\tilde{C}_{\text{new}}$ is ≤ 1, then we return the unique hypothesis. Otherwise, we continue to check if the model counter stops making progress by checking the old and new V sizes difference smaller than 1 in line 20. If stagnation is detected, we verify that if there exists an input x and two different hypotheses would have different outputs on x, and if this condition is checked with output UNSAT in line 23, then we conclude functional equivalence for all remaining trees and return any of the hypotheses. To guarantee that our algorithm always terminates, we add a sanity check in lines 28-31 that if the maximum round is reached, then we fail to find the unique hypothesis, where the maximum round is determined by the feature number. However, we never encounter this scenario in our experiments.

4 General CNF Encoding for Depth-d Decision Trees

We now present a general propositional encoding for the hypothesis space $\mathcal{H}_{d,n}$ of decision trees of maximum depth d over n binary features $x_1, \ldots, x_n \in \{0, 1\}$. This encoding supports exact SAT solving and approximate model counting with `ApproxMC`, enabling our active learning algorithm to prune the hypothesis space incrementally as oracle responses are received.

A full binary tree of depth d has:

$$N_{\text{int}} = 2^d - 1 \quad \text{internal nodes}, \qquad N_{\text{leaf}} = 2^d \quad \text{leaves}.$$

Lemma 1. *Let $\mathcal{H}_{d,n}$ be the class of full (complete) binary decision trees of depth d over n Boolean features, where*

- *each internal node selects* one *feature from the n features (selection may repeat across nodes),*
- *each leaf stores a Boolean output in $\{0, 1\}$.*

Then the number of distinct hypotheses in $\mathcal{H}_{d,n}$ is

$$|\mathcal{H}_{d,n}| \;=\; n^{\,2^d-1} \;\cdot\; 2^{\,2^d}.$$

The proof is shown in the Appendix.

4.1 Internal Node Feature Selection

Each internal node u selects exactly one feature to test. We introduce variables $S_{u,i} \quad (u = 1 \dots N_{\text{int}},\ i = 1 \dots n)$ representing node u tests feature x_i.

Constraint ϕ_{sel}. At least one feature must be chosen, and no two features may be selected simultaneously.

$$\phi_{\text{sel}} \ = \ \vee_{i=1}^{n} S_{u,i} \bigwedge \wedge_{1 \le i < j \le n} (\neg S_{u,i} \ \vee \ \neg S_{u,j}).$$

4.2 Leaf Labeling

Each leaf v has exactly one class from the label set $\mathcal{Y}$. We introduce variables: $L_{v,c}, c \in \mathcal{Y}$.

Constraint ϕ_{leaf}. Similarly as internal node featue selection,

$$\phi_{\text{sel}} \ = \ \vee_{c \in \mathcal{Y}} L_{v,c} \bigwedge \wedge_{c < c'} (\neg L_{v,c} \vee \neg L_{v,c'}).$$

4.3 Reachability of Nodes

For each input x in the set of evaluated examples, we introduce variables: $R_u(x)$, which means input x can reach node u.

Constraint ϕ_{root}. The root is always reachable: $\phi_{\text{root}} = R_r(x)$.

Propagation Constraints ϕ_{prop}. Suppose internal node u tests feature x_i. Then:

$$R_{\text{left}} \iff \big(R_u(x) \wedge x_i\big), \quad R_{\text{right}} \iff \big(R_u(x) \wedge \neg x_i\big).$$

Since the selected feature is itself encoded by $\{S_{u,i}\}_i$, we write:

$$\phi_{\text{prop-left}} \ = \ (\neg S_{u,i} \vee \neg R_u(x) \vee \neg x_i \vee R_{\text{left}(u)}(x)),$$

$$\phi_{\text{prop-right}} \ = \ (\neg S_{u,i} \vee \neg R_u(x) \vee x_i \vee R_{\text{right}(u)}(x)).$$

Full propagation constraint at node u:

$$\phi_{\text{prop}} = \bigwedge_{i=1}^{n} \big(\phi_{\text{prop-left}_i} \ \wedge \ \phi_{\text{prop-right}_i}\big).$$

4.4 Oracle Consistency Constraints

Whenever the membership oracle returns (x^*, y^*), every valid hypothesis must send x^* to a leaf that outputs y^*.

For each leaf v:

$$\phi_{\text{oracle}} = (\neg R_v(x^*) \vee L_{v,y^*}).$$

This constraint is added incrementally after each query, shrinking the remaining hypothesis space.

4.5 Putting Everything Together

The complete CNF encoding is:

$$\phi_{\text{all}} = \phi_{\text{select}} \ \wedge \ \phi_{\text{leaf}} \ \wedge \ \phi_{\text{root}} \ \wedge \ \phi_{\text{prop}} \ \wedge \ \phi_{\text{oracle}}$$

This single SAT instance represents the current space of hypotheses consistent with all oracle responses observed so far. `ApproxMC` is run over ϕ_{all} to estimate the remaining number of valid decision trees at each iteration of active learning.

5 Motivation Example

We now present a concrete worked example illustrating how our SAMCAL active learner incrementally eliminates incorrect decision trees until a unique hypothesis is identified. We restrict the hypothesis space to:

- a full binary decision tree of depth $d = 2$,
- $n = 3$ binary input features x_1, x_2, x_3,
- binary classification labels $\mathcal{Y} = \{0, 1\}$.

A depth-2 tree has $N_{\text{int}} = 2^2 - 1 = 3$ internal nodes, $N_{\text{leaf}} = 2^2 = 4$ leaves. Each internal node must choose exactly one of the three input features, and each leaf must select either output label 0 or 1. Thus, the theory permits: $|\mathcal{H}| = n^{N_{\text{int}}} \cdot 2^{N_{\text{leaf}}} = 3^3 \cdot 2^4 = 432$ candidate trees before any evidence is observed.

5.1 True Hidden Tree

For the running example, the oracle corresponds to a fixed hidden decision tree unknown to the learner:

$$Tree : \begin{cases} \text{Node 1 tests } x_3, & \\ \text{Node 2 tests } x_1, & \quad \text{Leaf outputs} = [1, 0, 1, 0]. \\ \text{Node 3 tests } x_2, & \end{cases}$$

The learner has no knowledge of this structure except through queries.

5.2 Incremental Learning Process

Before any queries, `ApproxMC` reports: $|\mathcal{H}| \approx 432$. Each round, the learner greedily select the input maximizing the expected reduction in the remaining models and then ask the oracle for output. Table 1 shows the actual execution from our tool. The first column shows the number of step and indicates the iterative round. The second and third columns show the selected input x^* and y^* from oracle. The last column shows how exactly the hypothesis space has reduced. The hypothesis steadily reduces from step one to step six, achieving near halving reduction. After seven queries, `ApproxMC` reports one remaining model, indicating that the learner has effectively identified a unique decision tree consistent with all observations without the need of checking functional equivalence. In this case, `ApproxMC` provides an exact and correct solution. We also notice that input (1,1,0) is not necessary to be queried to find this specific tree, indicating inputs selection is sensitive to ground truth during the learning process.

Step	**Selected Queried Input x^***	**Oracle Output y^***	**Approx. Remaining**
0	–	–	432
1	(0,0,0)	0	216
2	(1,1,1)	1	108
3	(0,0,1)	0	72
4	(0,1,1)	0	35
5	(1,0,0)	0	17
6	(1,0,1)	1	9
7	(0,1,0)	1	1

Table 1. Approximate model counts over successive active-learning rounds.

6 Experiments

We have implemented our method in a tool named `SAMCAL` written in Python with 2k lines of code, which builds on the ApproxMC 4.1.24 approximate model counter [7] and MiniSat 2.2 SAT solver [41] within PySAT [42]. The tool constructs a CNF encoding for the decision tree hypothesis space, with constraints for internal nodes, leaf outputs, and feature assignments. ApproxMC is used to estimate the number of remaining hypotheses after each query, guiding the active learning process by selecting queries that maximize hypothesis space reduction. When ApproxMC stagnates, `SAMCAL` invokes MiniSat to check for functional equivalence by verifying whether all remaining hypotheses compute the same Boolean function. If a functional collapse is detected (i.e., all hypotheses are functionally equivalent), the learning process terminates early, ensuring efficiency.

We evaluate our tool under the decision trees setting with depth = [2, 3, 4] and features = [3, 4, 5]. Judging from the parameters, it seems the tree is

small, however, when the depth is 4 with 5 features, the hypothesis space is $2 * 10^{15}$ (initial hypothesis size is shown in the last column in Table 2, growing at a speed more than exponential), and which is sufficient to demonstrate our tool's capacity. For each tree setting, we randomly generate one ground truth table consisting of node and feature correspondence and leaf boolean values. Our experiments were designed to answer the following questions:

- **RQ1**: Is our method effectively reduce the hypothesis space to identify the target decision tree?
- **RQ2**: How does our method scale when features and depth increase?
- **RQ3**: When stagnation occurs on ApproxMC's counting, does functional equivalence collapse occur?

The experiments were conducted on a computer with AMD Ryzen 5 5600X CPU and 32GB memory, running Ubuntu 20.04. In Table 2, we show the initialization of all trees' variables and clauses size in CNF formulas. Columns 1 and 2 shows the feature and depth numbers. The third and fourth column shows the variable and clauses size. We notice that when the feature number is the same, the CNF variables are doubling and clauses sizes are tripling as depth increases. When the depth number is the same, the CNF variables and clauses sizes are doubling as features increases. The depth is more dominant on clauses size than feature numbers.

Features	**Depth**	**CNF Vars**	**CNF Clauses**	**Hypothesis Size $\|\mathcal{H}\|$**
3	2	77	324	432
3	3	157	868	559,872
3	4	317	2,404	940,369,969,152
4	2	144	693	1,024
4	3	292	1,841	4,194,304
4	4	588	5,033	70,368,744,177,664
5	2	275	1,473	2,000
5	3	555	3,885	20,000,000
5	4	1,115	10,501	2,000,000,000,000,000

Table 2. Initial CNF Variables and Clauses Size on all Trees

6.1 Effectiveness - Results for Answering RQ1

To answer whether our method is effective to reduce the hypothesis and identify the target decision tree. We show in the hypothesis space drop trend for each tree in Figure 2a. The x-axis denotes the number of queries, and the y-axis denotes the size of hypothesis spaces on a log scale. Across all evaluated configurations (feature counts 3–4 and depths 2–4), our method consistently reduced the hypothesis space at an exponential rate before query 10. For deeper trees

such as n=4, d=4 and n=5, d=4, the initial hypothesis spaces were enormous, ranging from hundreds of thousands to several billion, yet the algorithm still achieved rapid reduction, typically stabilizing or collapsing within 8–12 queries. The deepest cases (d=4) started from exceedingly large spaces (up to 10^{15}), but still exhibited smooth monotone decay under our query strategy. Note in several cases, there is a sudden reduction benefiting from one query, e.g., in query 11 from tree n=4, d=4, there is a sudden space reduction over several magnitude. The reason is that this query is highly informative and remedies the non-optimality of previous queries. In four cases n=4/5, d=3/4), we can see a short flat curve, which indicates approximation of the models making little progress. It still makes progress; otherwise, the curve would terminate earlier. However, this means effectiveness of our method and this behavior will be addressed in Section 6.3. Overall, these results demonstrate that our selection strategy is highly effective at driving fast version-space collapse regardless of tree depth or feature count.

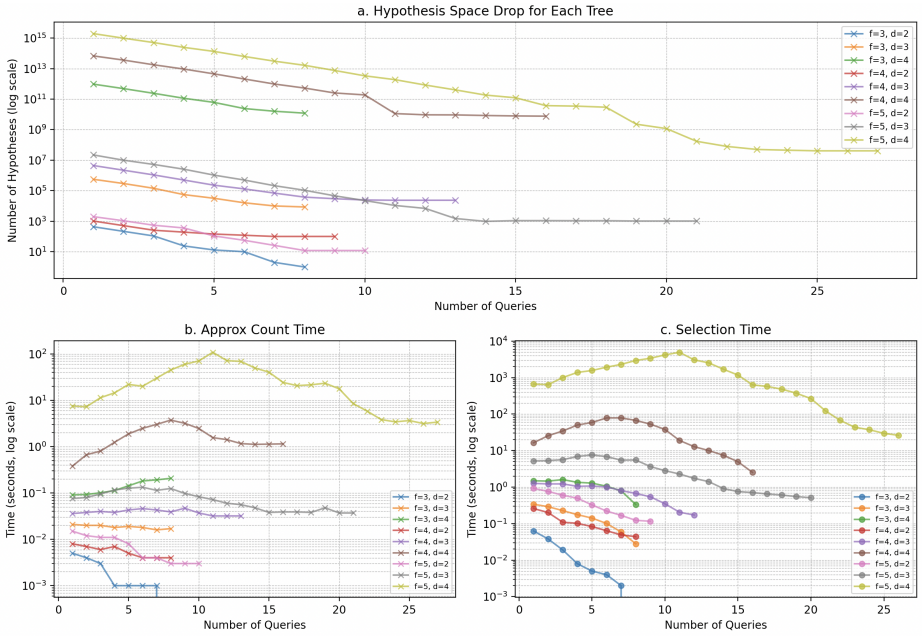

Fig. 2. Hypothesis Space Drop, Count, and Selection Time Trend

6.2 Scalability - Results for Answering RQ2

We show our timing results in Figure 2b and 2c. The x-axis denotes the number of queries, and the y-axis denotes the time spent on a log scale. Figure 2b shows

the initial and following approximation of the number of remaining models after adding input-output pair constraint from each query. The cost of querying scales with both the feature count and tree depth, exhibiting predictable growth as the hypothesis space becomes larger. For small and moderate configurations (e.g., n=3/4, d=2/3), both the ApproxMC counting time and the query-selection time remain well under a second. As depth increases to d=4, the per-query cost grows substantially, which is as expected from the exponential hypothesis space, yet remains well-behaved: for f=4 the peak selection time reaches only a few seconds, and for n=5 the most demanding configuration (n=5, d=4) shows a rapid rise to the worst-case of 4k seconds per selection round. Importantly, even in this extreme setting the version space reduces steadily, causing subsequent ApproxMC and selection times to decrease quickly. This behavior indicates that although our method naturally inherits exponential scaling from the underlying combinatorial complexity, the active-learning process benefits from fast hypothesis elimination, keeping the overall runtime manageable and demonstrating practical scalability across all evaluated tree sizes.

6.3 Functional Equivalence Follows Stagnation - Results for Answering RQ3

Tree	Step	# Before	# After	Stagnate	Func Equi	Solve Time	Reduction
n4_d2	7→8	100	1	yes	yes	$<$ 1s	50% (8)
n4_d3	11→12	23,552	1	yes	yes	$<$ 1s	25% (5)
n5_d2	8→9	12	1	yes	yes	$<$ 1s	72% (23)
n5_d3	19→20	1,024	1	yes	yes	$<$ 1s	38% (12)
n5_d4	25→26	40,894,464	1	yes	yes	$<$ 1s	19% (6)

Table 3. Stagnation & Functional Collapse (RQ3)

Table 3 shows the results of detecting stagnation in ApproxMC and verifying functional equivalence across different experimental settings. Column 1 shows the tree topology. For each experiment, we identify the step in Column 2, showing at which stage that the approximate count of remaining hypotheses stops decreasing, indicating stagnation. The before and after approximating model numbers are shown in Column 3 and 4. At these steps, the functional equivalence of the remaining hypotheses is verified using a SAT solver, confirming that the hypothesis space has collapsed to a unique solution. Column 5 and 6 indicates if a stagnation is found and if the functional equivalence is verified.

The experiments show that in all cases, the SAT check completes in less than 1 second, demonstrating the efficiency of the functional equivalence check. The count before stagnation varies depending on the problem size (e.g., 100 hypotheses for n=4, d=2 and 23,552 hypotheses for n=4, d=3, but the method consistently reduces the hypothesis space to a single solution in very few steps.

This indicates that our active learning method is effective at quickly identifying the correct Boolean function, with functional collapse detected at each step of stagnation, confirming the validity of the approach. The last column shows the percentage (the number of queries) our method reduced from the all features permutation queries (e.g., 4 features which require $2^4 = 16$ queries). For all trees that stagnate during the learning process, at least 19%/6 queries and at most 72%/23 queries were saved. They also demonstrate our tool's effectiveness on query number optimization.

7 Discussion

In this section, we discuss theoretical and practical considerations for scaling to larger trees, and the limitations of our methods.

Scalability Considerations: The full hypothesis space for a depth-d decision tree with n Boolean features has size $n^{(2^d-1)} \cdot 2^{2^d}$, where the first factor enumerates feature assignments for internal nodes, and the second assigns outputs to leaves. This space grows exponentially in d, making explicit enumeration infeasible beyond shallow depths. Our approach remains tractable on our experimental settings because the entire space is represented symbolically using CNF, avoiding explicit enumeration. `ApproxMC` provides approximate counts in time sublinear in $|\mathcal{H}|$, using hashing-based universal model counting [16]. Each highly selected query reduces the version space by a factor close to one half evidenced in our experiments. One might also expect each membership query to eliminate roughly half of the candidate trees, since decision trees partition the input space at each node. In reality, a query constrains only the leaves and internal nodes along its input path, leaving many other hypotheses consistent. The fraction of eliminated trees varies per query and is rarely exactly 50%. By using approximate model counting, our method quantitatively estimates the version-space reduction for each candidate query efficiently. It is worthwhile to note that `ApproxMC` may be unable to effectively scale up if the function space is inherently hard to reduce.

Limitations: Several limitations of our method should be acknowledged. Since the runtime of our algorithm scales with the cost of approximate model counting, which can still be nontrivial for large CNFs. We can handle as many as 10k CNF clauses with over 1k variables. Furthermore, when the internal nodes support more than binary decisions, the formula complexity would also grow significantly. There may be a better strategy to balance the optimality for each round and time. For example, a parallel algorithm can be leveraged in the query selection procedure to speed up 10x with 10 threads, which can approximate the models for different inputs in parallel. However, our focus here is to find the approximately optimal choice iteratively, and we leave the parallel aspect for future directions. Another threat is `ApproxMC`-based estimates may occasionally select suboptimal queries if estimation error skews comparison. Our method also does not provide theoretical query complexity guarantees, where no worst-bounds results are achieved. Finally, our evaluation assumes a noise-free oracle.

In realistic settings with noisy or adversarial labels [15], space elimination may become inconsistent, and the current SAT-based mechanism offers no robustness guarantees. Despite these constraints, the empirical behavior observed in Section 6 confirms that the algorithm remains practical and robust for medium-scaled decision trees.

8 Related Work

Classical exact active learning was introduced by Angluin [5, 6], who studied learning in the exact identification setting with membership queries. Decision-tree inference from queries has also been explored in theory [25, 46], including bounds on optimal decision tree learning and information gain heuristics. Our contribution differs in that we do not operate over an explicit enumeration of tree hypotheses; instead, we maintain a *symbolic* representation in CNF and use constraints to guarantee consistency.

Model counting has a long history in formal verification and probabilistic reasoning. Exact model counters such as SHARPSAT [43] achieve high performance in practice but still remain worst-case exponential. Approximate counters such as `ApproxMC` [17] provide provable (ε, δ) guarantees using universal hashing. The idea of using approximate model counts to guide search has appeared in domains such as probabilistic inference [18, 22], but, to our knowledge, has not previously been used to *drive optimal active query selection* in formal learning frameworks.

SAT and SMT-based learning from examples has been explored in inductive program synthesis [4, 36], repair [27, 29], and invariant generation [24, 28]. These works typically add constraints incrementally as new counter examples learned, but they rely either on heuristic search or program abstract domain rather than numerical estimation of version space reduction. In contrast, our system quantifies the remaining hypothesis space volume after each candidate query, allowing a principled halving strategy analogous to classical active learning but with no need to enumerate the version space explicitly.

9 Conclusion

We introduced a new framework for active learning in which the hypothesis space is represented symbolically in CNF and the expected information gain of each candidate query is estimated using approximate model counting. This enables a learning strategy without enumerating the hypothesis space, even when the number of possible functions is combinatorially huge. We demonstrated this approach using various numbers of depth and features of unknown decision trees as targets, encoding them in Boolean variables. `ApproxMC` was used to estimate, for each possible query input, the approximate number of hypotheses consistent with each oracle response. The input minimizing the estimated remaining version space was chosen automatically, and incremental solving strategy enforced consistency with observed answers. We showed that when `ApproxMC` stagnates, a functional collapse check is performed to verify a functional equivalent tree is

found. For future work, we plan to extend our method to more applications and discover more efficient oracle input selection strategies.

Acknowledgments Zunchen Huang and Chenglu Jin are (partially) supported by project CiCS of the research programme Gravitation, which is (partly) financed by the Dutch Research Council (NWO) under the grant 024.006.037. We thank Chao Yin, Marten van Dijk, and Fabio Massacci for their discussions on function recovery on gate-hiding garbled circuits, providing some insights on this work.

References

1. Aasi, E., Cai, M., Vasile, C.I., Belta, C.: Time-incremental learning of temporal logic classifiers using decision trees. In: Matni, N., Morari, M., Pappas, G.J. (eds.) Learning for Dynamics and Control Conference, L4DC 2023, 15-16 June 2023, Philadelphia, PA, USA. Proceedings of Machine Learning Research, vol. 211, pp. 547–559. PMLR (2023), https://proceedings.mlr.press/v211/aasi23a.html
2. Aasi, E., Vasile, C.I., Bahreinian, M., Belta, C.: Inferring temporal logic properties from data using boosted decision trees. CoRR **abs/2105.11508** (2021), https://arxiv.org/abs/2105.11508
3. Aichernig, B.K., Tappler, M.: Probabilistic black-box reachability checking. In: Lahiri, S.K., Reger, G. (eds.) Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10548, pp. 50–67. Springer (2017). https://doi.org/10.1007/978-3-319-67531-2_4, https://doi.org/10.1007/978-3-319-67531-2_4
4. Alur, R., Bodík, R., Juniwal, G., Martin, M.M.K., Raghothaman, M., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.: Syntax-guided synthesis. In: Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013. pp. 1–8. IEEE (2013), https://ieeexplore.ieee.org/document/6679385/
5. Angluin, D.: Inductive inference of formal languages from positive data. Inf. Control. **45**(2), 117–135 (1980). https://doi.org/10.1016/S0019-9958(80)90285-5, https://doi.org/10.1016/S0019-9958(80)90285-5
6. Angluin, D., Smith, C.H.: Inductive inference: Theory and methods. ACM Comput. Surv. **15**(3), 237–269 (1983). https://doi.org/10.1145/356914.356918, https://doi.org/10.1145/356914.356918
7. ApproxMC: Approxmc6: Approximate model counter (2024), https://github.com/meelgroup/approxmc/releases/tag/4.1.24
8. Balcan, M., Feldman, V.: Statistical active learning algorithms. In: Burges, C.J.C., Bottou, L., Ghahramani, Z., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States. pp. 1295–1303 (2013), https://dl.acm.org/doi/10.5555/2999611.2999756
9. Bau, J., Bursztein, E., Gupta, D., Mitchell, J.C.: State of the art: Automated black-box web application vulnerability testing. In: 31st IEEE Symposium on Security and Privacy, SP 2010, 16-19 May 2010, Berleley/Oakland, California, USA. pp. 332–345. IEEE Computer Society (2010). https://doi.org/10.1109/SP.2010.27, https://doi.org/10.1109/SP.2010.27

10. Bombara, G., Belta, C.: Offline and online learning of signal temporal logic formulae using decision trees. ACM Trans. Cyber Phys. Syst. **5**(3), 22:1–22:23 (2021). https://doi.org/10.1145/3433994, https://doi.org/10.1145/3433994
11. Bombara, G., Vasile, C.I., Penedo, F., Yasuoka, H., Belta, C.: A decision tree approach to data classification using signal temporal logic. In: Abate, A., Fainekos, G. (eds.) Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, Vienna, Austria, April 12-14, 2016. pp. 1–10. ACM (2016). https://doi.org/10.1145/2883817.2883843, https://doi.org/10.1145/2883817.2883843
12. Bowring, J.F., Rehg, J.M., Harrold, M.J.: Active learning for automatic classification of software behavior. In: Avrunin, G.S., Rothermel, G. (eds.) Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2004, Boston, Massachusetts, USA, July 11-14, 2004. pp. 195–205. ACM (2004). https://doi.org/10.1145/1007512.1007539, https://doi.org/10.1145/1007512.1007539
13. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth (1984)
14. Brunello, A., Sciavicco, G., Stan, I.E.: Interval temporal logic decision tree learning. In: Calimeri, F., Leone, N., Manna, M. (eds.) Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11468, pp. 778–793. Springer (2019). https://doi.org/10.1007/978-3-030-19570-0_50, https://doi.org/10.1007/978-3-030-19570-0_50
15. Cesa-Bianchi, N., Gentile, C., Vitale, F., Zappella, G.: Active learning on trees and graphs. In: Kalai, A.T., Mohri, M. (eds.) COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010. pp. 320–332. Omnipress (2010), http://colt2010.haifa.il.ibm.com/papers/COLT2010proceedings.pdf#page=328
16. Chakraborty, S., Meel, K.S., Mistry, R., Vardi, M.Y.: Approximate probabilistic inference via word-level counting. In: Schuurmans, D., Wellman, M.P. (eds.) Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. pp. 3218–3224. AAAI Press (2016). https://doi.org/10.1609/AAAI.V30I1.10416, https://doi.org/10.1609/aaai.v30i1.10416
17. Chakraborty, S., Meel, K.S., Vardi, M.Y.: A scalable approximate model counter. In: Schulte, C. (ed.) Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8124, pp. 200–216. Springer (2013). https://doi.org/10.1007/978-3-642-40627-0_18, https://doi.org/10.1007/978-3-642-40627-0_18
18. Chavira, M., Darwiche, A.: On probabilistic inference by weighted model counting. Artif. Intell. **172**(6-7), 772–799 (2008). https://doi.org/10.1016/J.ARTINT.2007.11.002, https://doi.org/10.1016/j.artint.2007.11.002
19. Chen, J., Cousot, P.: A binary decision tree abstract domain functor. In: Blazy, S., Jensen, T.P. (eds.) Static Analysis - 22nd International Symposium, SAS 2015, Saint-Malo, France, September 9-11, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9291, pp. 36–53. Springer (2015). https://doi.org/10.1007/978-3-662-48288-9_3, https://doi.org/10.1007/978-3-662-48288-9_3
20. Chen, Y., Wang, B.: Learning boolean functions incrementally. In: Madhusudan, P., Seshia, S.A. (eds.) Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings. Lecture Notes in Com-

puter Science, vol. 7358, pp. 55–70. Springer (2012). https://doi.org/10.1007/978-3-642-31424-7_10, https://doi.org/10.1007/978-3-642-31424-7_10

21. Cousot, P., Cousot, R., Mauborgne, L.: A scalable segmented decision tree abstract domain. In: Manna, Z., Peled, D.A. (eds.) Time for Verification, Essays in Memory of Amir Pnueli. Lecture Notes in Computer Science, vol. 6200, pp. 72–95. Springer (2010). https://doi.org/10.1007/978-3-642-13754-9_5, https://doi.org/10.1007/978-3-642-13754-9_5
22. Dubray, A., Schaus, P., Nijssen, S.: Probabilistic inference by projected weighted model counting on horn clauses. In: Yap, R.H.C. (ed.) 29th International Conference on Principles and Practice of Constraint Programming, CP 2023, Toronto, Canada, August 27-31, 2023. LIPIcs, vol. 280, pp. 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.CP.2023.15, https://doi.org/10.4230/LIPIcs.CP.2023.15
23. Francis, P., Leon, D., Minch, M., Podgurski, A.: Tree-based methods for classifying software failures. In: 15th International Symposium on Software Reliability Engineering (ISSRE 2004), 2-5 November 2004, Saint-Malo, Bretagne, France. pp. 451–462. IEEE Computer Society (2004). https://doi.org/10.1109/ISSRE.2004.43, https://doi.org/10.1109/ISSRE.2004.43
24. Garg, P., Neider, D., Madhusudan, P., Roth, D.: Learning invariants using decision trees and implication counterexamples. In: Bodík, R., Majumdar, R. (eds.) Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016. pp. 499–512. ACM (2016). https://doi.org/10.1145/2837614.2837664, https://doi.org/10.1145/2837614.2837664
25. Gupta, A., Nagarajan, V., Ravi, R.: Approximation algorithms for optimal decision trees and adaptive TSP problems. Math. Oper. Res. **42**(3), 876–896 (2017). https://doi.org/10.1287/MOOR.2016.0831, https://doi.org/10.1287/moor.2016.0831
26. Huang, Y., Jiang, J.R.: Circuit learning: From decision trees to decision graphs. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **42**(11), 3985–3996 (2023). https://doi.org/10.1109/TCAD.2023.3258747, https://doi.org/10.1109/TCAD.2023.3258747
27. Huang, Z., Wang, C.: Constraint based program repair for persistent memory bugs. In: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024. pp. 91:1–91:12. ACM (2024). https://doi.org/10.1145/3597503.3639204, https://doi.org/10.1145/3597503.3639204
28. Krishna, S., Puhrsch, C., Wies, T.: Learning invariants using decision trees. CoRR (2015), http://arxiv.org/abs/1501.04725
29. Le Goues, C., Holtschulte, N.J., Smith, E.K., Brun, Y., Devanbu, P.T., Forrest, S., Weimer, W.: The manybugs and introclass benchmarks for automated repair of C programs. IEEE Trans. Software Eng. **41**(12), 1236–1256 (2015). https://doi.org/10.1109/TSE.2015.2454513, https://doi.org/10.1109/TSE.2015.2454513
30. Li, X., Xue, Y.: BLOCK: a black-box approach for detection of state violation attacks towards web applications. In: Zakon, R.H., McDermott, J.P., Locasto, M.E. (eds.) Twenty-Seventh Annual Computer Security Applications Conference, ACSAC 2011, Orlando, FL, USA, 5-9 December 2011. pp. 247–256. ACM (2011). https://doi.org/10.1145/2076732.2076767, https://doi.org/10.1145/2076732.2076767

31. Liang, K., Cardona, G.A., Kamale, D., Vasile, C.: Learning optimal signal temporal logic decision trees for classification: A max-flow MILP formulation. In: 63rd IEEE Conference on Decision and Control, CDC 2024, Milan, Italy, December 16-19, 2024. pp. 8332–8337. IEEE (2024). https://doi.org/10.1109/CDC56724.2024.10886055, https://doi.org/10.1109/CDC56724.2024.10886055
32. Mariani, L., Pezzè, M., Riganelli, O., Santoro, M.: Autoblacktest: a tool for automatic black-box testing. In: Taylor, R.N., Gall, H.C., Medvidovic, N. (eds.) Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011. pp. 1013–1015. ACM (2011). https://doi.org/10.1145/1985793.1985979, https://doi.org/10.1145/1985793.1985979
33. McGraw, G., Potter, B.: Software security testing. IEEE Secur. Priv. **2**(5), 81–85 (2004). https://doi.org/10.1109/MSP.2004.84, https://doi.org/10.1109/MSP.2004.84
34. Peled, D.A., Vardi, M.Y., Yannakakis, M.: Black box checking. In: Wu, J., Chanson, S.T., Gao, Q. (eds.) Formal Methods for Protocol Engineering and Distributed Systems, FORTE XII / PSTV XIX'99, IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX), October 5-8, 1999, Beijing, China. IFIP Conference Proceedings, vol. 156, pp. 225–240. Kluwer (1999)
35. Pellegrino, G., Balzarotti, D.: Toward black-box detection of logic flaws in web applications. In: 21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014. The Internet Society (2014), https://www.ndss-symposium.org/ndss2014/toward-black-box-detection-logic-flaws-web-applications
36. Polozov, O., Gulwani, S.: Flashmeta: a framework for inductive program synthesis. In: Aldrich, J., Eugster, P. (eds.) Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, part of SPLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015. pp. 107–126. ACM (2015). https://doi.org/10.1145/2814270.2814310, https://doi.org/10.1145/2814270.2814310
37. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986). https://doi.org/10.1023/A:1022643204877, https://doi.org/10.1023/A:1022643204877
38. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: Alur, R., Peled, D.A. (eds.) Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3114, pp. 202–215. Springer (2004). https://doi.org/10.1007/978-3-540-27813-9_16, https://doi.org/10.1007/978-3-540-27813-9_16
39. Shijubo, J., Waga, M., Suenaga, K.: Efficient black-box checking via model checking with strengthened specifications. In: Feng, L., Fisman, D. (eds.) Runtime Verification - 21st International Conference, RV 2021, Virtual Event, October 11-14, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12974, pp. 100–120. Springer (2021). https://doi.org/10.1007/978-3-030-88494-9_6, https://doi.org/10.1007/978-3-030-88494-9_6
40. Sloan, R.H., Szörényi, B., Turán, G., Crama, Y., Hammer, P.L.: Learning boolean functions with queries. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering, pp. 221–256. Cam-

bridge University Press (2010). https://doi.org/10.1017/CBO9780511780448.010, https://doi.org/10.1017/cbo9780511780448.010
41. Sorensson, N., Claessen, K.: A minimalistic and high-performance sat solver (2010), https://github.com/niklasso/minisat/tree/releases/2.2.0
42. Stoneback, R., Klenzing, J., Burrell, A., Spence, C., Depew, M., Hargrave, N., Smith, J., von Bose, V., Pembroke, A., Iyer, G., Luis, S.: Python satellite data analysis toolkit (pysat) vx.y.z (2021). https://doi.org/10.5281/zenodo.1199703, https://doi.org/10.5281/zenodo.1199703
43. Thurley, M.: sharpsat - counting models with advanced component caching and implicit BCP. In: Biere, A., Gomes, C.P. (eds.) Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4121, pp. 424–429. Springer (2006). https://doi.org/10.1007/11814948_38, https://doi.org/10.1007/11814948_38
44. Urban, C., Miné, A.: A decision tree abstract domain for proving conditional termination. In: Müller-Olm, M., Seidl, H. (eds.) Static Analysis - 21st International Symposium, SAS 2014, Munich, Germany, September 11-13, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8723, pp. 302–318. Springer (2014). https://doi.org/10.1007/978-3-319-10936-7_19, https://doi.org/10.1007/978-3-319-10936-7_19
45. Xie, T., Notkin, D.: Checking inside the black box: Regression testing by comparing value spectra. IEEE Trans. Software Eng. **31**(10), 869–883 (2005). https://doi.org/10.1109/TSE.2005.107, https://doi.org/10.1109/TSE.2005.107
46. Zhuo, Z., Nagarajan, V.: A simple approximation algorithm for optimal decision tree. Oper. Res. Lett. **64**, 107370 (2026). https://doi.org/10.1016/J.ORL.2025.107370, https://doi.org/10.1016/j.orl.2025.107370

Appendix

Proof for Lemma 1:

Proof. A full binary tree of depth d (root at depth 0) has exactly

$$N_{\text{leaf}} = 2^d$$

leaves and

$$N_{\text{int}} = 2^d - 1$$

internal (non-leaf) nodes. This is standard: each level k ($0 \le k < d$) contains 2^k internal nodes, and summing yields $\sum_{k=0}^{d-1} 2^k = 2^d - 1$.

A hypothesis in $\mathcal{H}_{d,n}$ is fully specified by two independent choices:

1. For every internal node (there are N_{int} of them) choose one feature from the n available features. Since selections are independent across nodes and repetition is allowed, the number of ways to assign features to all internal nodes is
$$n^{N_{\text{int}}} = n^{2^d-1}$$
2. For every leaf (there are N_{leaf} of them) choose a Boolean output in $\{0, 1\}$. The number of ways to label all leaves is
$$2^{N_{\text{leaf}}} = 2^{2^d}.$$

Because these two choices are independent (feature selections do not constrain leaf labels and vice versa), the total number of hypotheses equals the product of the two counts:

$$|\mathcal{H}_{d,n}| \;=\; n^{N_{\text{int}}} \cdot 2^{N_{\text{leaf}}} \;=\; n^{2^d-1} \cdot 2^{2^d}.$$

This completes the proof. □