LOGIC-KAN: COMPACT BOOLEAN PROGRAM INDUCTION THROUGH DIFFERENTIABLE LEARNING AND STRUCTURAL EDITING

Anonymous authors
Paper under double-blind review

ABSTRACT

Boolean rule induction seeks to recover an executable logical program from labeled binary examples. Yet high held-out accuracy can conceal disagreements with the target rule elsewhere in the Boolean domain, and learning with a fixed clause budget can produce unnecessarily large programs. We propose Logic-KAN, a differentiable model that assembles positive and negated literals into shared conjunctive clauses and combines them through OR and XOR heads. Continuous parameters learn which literals and clauses to select and how each head combines its clauses, while a thresholded version exposes the current rule during training. Using only labeled training examples, Logic-KAN scores local edits that add, refine, replace, or remove clauses, adapting its structure as gradient learning proceeds. After training, the discrete model is compiled into an executable symbolic program whose clauses and operators can be inspected directly. Separate checks compare the exported program with the discrete model and, when a target formula is available, use SAT to test equivalence over the full Boolean domain. On synthetic DNF and mixed OR/XOR tasks, SAT finds counterexamples in some runs with perfect held-out accuracy. Compared with existing models, Logic-KAN produces smaller programs on both task families and achieves better exact recovery on DNF.

1 INTRODUCTION

Boolean rule induction learns an executable program from labeled binary examples. Given $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$ with $x_i \in \{0,1\}^d$, the learner must select relevant literals, form clauses, and combine those clauses into a Boolean function. An explicit program shows which conditions determine a prediction and can be run without the training model. When a reference formula is available, the program can also be tested against it on every Boolean assignment.

Yet accurate predictions on sampled inputs do not establish that the exported rule is compact or agrees with the target throughout the Boolean domain. Held-out accuracy measures the rule on a finite set; exact recovery requires $P(x) = f^\star(x)$ for every $x \in \{0,1\}^d$. In the final Conjunction experiment, one learned program labels every held-out example correctly, but SAT finds an input on which it disagrees with the generating rule. The program obtained by thresholding continuous parameters may also behave differently from the trained predictor. These differences matter when the learned rule itself is the object to be inspected or executed.

Differentiable logic models learn trainable rules or relaxed gate operations through gradient descent (Reimann et al., 2022; Petersen et al., 2022). Pruning and thresholding extract symbolic rules (Cingillioglu & Russo, 2021), while gradient grafting trains discrete rules using continuous gradients (Wang et al., 2021). However, fitting the continuous predictor does not ensure that its discrete rule preserves those predictions or has a compact structure. This motivates using the hard rule's errors and structural cost to guide learning directly.

Discrete rule search explicitly balances empirical error and complexity (Lawless et al., 2023), but even a compact, sample-consistent rule may disagree with the target on unobserved inputs. Here, the target formula is withheld from learning: examples guide structural choices, and target equivalence is checked after program selection.

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.

Learning that structure presents a combinatorial problem. A clause can include each variable positively, negatively, or not at all. A change to one clause may affect several OR or XOR heads that share it, so the effect of an edit cannot be determined from the clause alone. Gradients can tune continuous parameters, but they do not directly decide when to add, refine, or remove an active clause. A fixed allocation supplies capacity, yet may leave redundant terms in the final program. Structural changes need to account for both the program's predictions on observed examples and its size.

Logic-KAN addresses this problem by making a Boolean program the shared object of continuous and discrete optimization. Its input-to-clause edges learn one of three unary functions, x_j, $1 - x_j$, or 1, analogous to edge-wise function selection in a KAN but specialized to literal choice. Products form shared conjunctions; learned memberships and OR/XOR operators compose them into a bounded program. The hard state interprets these same parameters as Boolean rules that are edited during training.

Structural candidates are scored on hard Boolean predictions and explicit program cost. Because the fixed outer OR can hide whether one head corrected an overlapping error, Logic-KAN evaluates the affected subprogram locally; we derive the precise masking relationship between this local score and the full-program risk. Gradient updates then adapt the continuous parameters to the edited structure. After validation checkpoint selection, conservative empirical minimization and guarded data-derived program candidates complete selection within the same grammar. A label-free compiler exports the final state, after which soft–hard fidelity, compiler fidelity, and target equivalence are evaluated separately. Target formulas and SAT counterexamples never enter training or model selection.

Our contributions are threefold:

- We introduce a Boolean KAN representation whose edge functions select signed literals or omission, whose shared conjunctions feed learned OR/XOR heads, and whose hard state is an executable program in the same grammar as the differentiable model.
- We combine gradient learning with bounded structural editing and derive how the outer OR masks an edit's contribution to full-program risk. The resulting local objective supplies a correction signal for overlapping heads while a zero-error guard protects already fitted training examples.
- We separate discretization fidelity, compiler fidelity, and SAT-based target equivalence. Across the full catalogue, the method recovers multiple target families and exactly matches the generating signed-clause set in all 155 SAT-exact Random-DNF runs.

2 Related Work

Logic-KAN studies Boolean function learning from labeled examples, with an emphasis on executable rules and functional correctness. The closest work concerns Boolean rule induction, differentiable structure learning, and symbolic extraction.

Boolean rule induction. Boolean compressed sensing formulates the recovery of individual conjunctive or disjunctive rules as group testing and establishes exact recovery under conditions on the sensing matrix (Malioutov & Varshney, 2013). CORELS searches regularized rule lists and certifies optimality for its empirical objective (Angelino et al., 2018). Such optimality is distinct from equivalence to a target function over the full input domain. Boolean Rule Column Generation trades empirical error against DNF or CNF complexity (Dash et al., 2018), while BoolXAI searches expressive Boolean formulas through local modifications (Kadioglu et al., 2025). Logic-KAN uses a bounded circuit of signed conjunctions shared across OR/XOR heads. Unlike direct search over rule lists or formulas, it combines gradient training with discrete edits to this circuit.

Differentiable structure learning. Continuous relaxations also support rule learning in inductive logic programming and knowledge-base reasoning (Evans & Grefenstette, 2018; Yang et al., 2017). DR-Net learns conjunctions with positive and negative feature associations and uses sparsity regularization to control rule complexity (Qiao et al., 2021). Mat_DNF gives matrix parameters explicit literal and conjunction-selection semantics and studies target-rule recovery from incomplete observations (Sato & Inoue, 2023). The Rule-based Representation Learner uses gradient grafting to optimize discrete logical representations with gradients from a continuous model (Wang et al.,

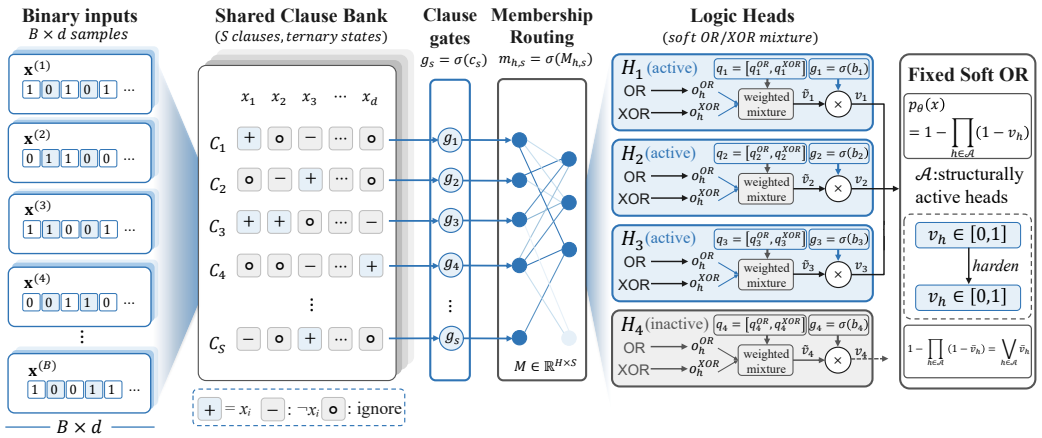


Figure 1: **Logic-KAN architecture.** Binary inputs form shared conjunctive clauses through ternary literal choices. Learned gates and memberships route these clauses to OR/XOR heads, whose active outputs are combined by a fixed soft OR.

2021). Deep Differentiable Logic Gate Networks learn distributions over Boolean operators on a fixed connectivity pattern before discretization (Petersen et al., 2022). Subsequent work improves gate parameterization (Rüttgers et al., 2026) and learns sparse connectivity (Fojcik et al., 2026). These methods already provide interpretable parameterizations and mechanisms for changing discrete rules or connections. Logic-KAN interleaves gradient updates with cost-aware edits to the hard circuit during training. Candidate edits can add, remove, or rewrite clauses, alter clause–head memberships, and switch head operators. We score these edits on training examples using changes in class-balanced local prediction error and explicit costs for literals, memberships, and heads.

Symbolic extraction and fidelity. Kolmogorov–Arnold Networks learn univariate edge functions (Liu et al., 2025). Logic-KAN restricts these edge functions to positive literals, negative literals, and omission, giving each discrete choice a Boolean meaning. In neural rule learning, pix2rule extracts symbolic rules through pruning and thresholding (Cingillioglu & Russo, 2021). Splitting entangled conjunctions can reduce the predictive degradation caused by translating neural DNF models into logic (Baugh et al., 2025). For logic gate networks, Gumbel noise and straight-through estimation reduce the discrepancy between relaxed training and discrete inference (Yousefi et al., 2025). SATNet embeds a differentiable satisfiability solver in a neural network (Wang et al., 2019); subsequent work decodes its learned parameters into weighted MaxSAT rules and checks equivalence to reference rules (Li et al., 2023). Logic-KAN checks whether discretization and compilation preserve predictions, and separately verifies whether the exported program is equivalent to the known target over all Boolean inputs.

3 Method

Logic-KAN treats a Boolean program as an object that can be optimized in two compatible views. The soft view supplies gradients over literal functions, clause gates, memberships, and head operators. The hard view interprets those same parameters as signed conjunctions connected to OR/XOR heads and supports explicit structural edits. Both views use at most S clause slots and $H_{\max}$ heads under a fixed outer OR.

During training, gradient updates alternate with bounded edits to clauses, memberships, and head operators. After validation checkpoint selection, empirical minimization and guarded joint program selection determine the final hard state. A label-free compiler exports this state, and independent checks assess soft–hard agreement, export fidelity, and target equivalence. The shared representation is shown in Figure 1. Appendix B gives the training and implementation sequence, while Appendix B.2 specifies candidate construction.

3.1 DIFFERENTIABLE BOOLEAN PROGRAM

Let $\mathbf{x} \in \{0,1\}^d$. For every slot s and feature j, logits $\boldsymbol{\ell}_{sj} = (\ell^+_{sj}, \ell^-_{sj}, \ell^0_{sj})$ describe three logical states. At temperature τ,

$$\boldsymbol{\alpha}_{sj} = \mathrm{softmax}(\boldsymbol{\ell}_{sj}/\tau), \qquad \phi_{sj}(x_j) = \alpha^+_{sj} x_j + \alpha^-_{sj}(1 - x_j) + \alpha^0_{sj}. \tag{1}$$

The third state is the constant function 1, so it omits the feature under conjunction. A learned gate $g_s = \sigma(\gamma_s)$ scales clause s:

$$c_s(\mathbf{x}) = g_s \prod_{j=1}^{d} \phi_{sj}(x_j). \tag{2}$$

The implementation evaluates this product in the log domain; numerical clipping is specified in Appendix B.1.

A membership gate $m_{hs} = \sigma(\mu_{hs})$ routes clause s to head h, giving $u_{hs}(\mathbf{x}) = m_{hs} c_s(\mathbf{x})$. We suppress the input argument below. Each head computes relaxed OR and XOR in parallel,

$$o_h = 1 - \prod_s (1 - u_{hs}), \qquad r_h = \frac{1 - \prod_s (1 - 2u_{hs})}{2}. \tag{3}$$

If $\boldsymbol{\pi}_h = \mathrm{softmax}(\boldsymbol{\omega}_h/\tau)$ is the learned distribution over OR and XOR and $b_h = \sigma(\beta_h)$ is the head gate, then

$$v_h = b_h(\pi_{h,\mathrm{OR}} o_h + \pi_{h,\mathrm{XOR}} r_h), \qquad p_\theta(\mathbf{x}) = 1 - \prod_h (1 - v_h). \tag{4}$$

Products in Equations (3) and (4) run only over structurally active slots and heads. Appendix A.2 proves that the one-hot, binary-gated limit of these operators evaluates the corresponding hard Boolean program.

3.2 HARD STATE AND PARAMETER LEARNING

The same parameters define a discrete program state that the structural editor reads and modifies during training. This hard state connects continuous optimization to rule modification and later compilation. Operator states use argmax. Literal states use argmax followed by an enforced per-clause cap $K = 6$. Clause, membership, and head gates are retained when their sigmoid value is at least $1/2$, subject to the structural masks. The resulting clause is

$$\bar{c}_s(\mathbf{x}) = \bigwedge_{j : z_{sj} = +} x_j \ \wedge \bigwedge_{j : z_{sj} = -} \neg x_j, \qquad z_{sj} = \left[\Pi_K(\tilde{\mathbf{z}}_s; \boldsymbol{\ell}_s)\right]_j. \tag{5}$$

Here $\tilde{z}_{sj} = \arg\max_{k \in \{+,-,0\}} \ell^k_{sj}$ and $\boldsymbol{\ell}_s$ contains the logits for all features in slot s. The projection Π_K uses these logits to retain at most K literals and maps the others to omission. An empty conjunction is true; each retained head applies its selected OR or XOR to retained clauses, and a fixed OR combines the heads. The hard executor evaluates this discrete structure directly. Projection margins, tie breaking, and empty-structure conventions are given in Appendix A.1.

Mini-batch learning minimizes

$$\mathcal{L}_{\mathrm{base}} = \mathrm{BCE}(p_\theta, y) + \eta_t \big(\lambda_{\mathrm{ent}} \mathcal{R}_{\mathrm{ent}} + \lambda_{\mathrm{gate}} \mathcal{R}_{\mathrm{gate}} + \lambda_{\mathrm{lit}} \mathcal{R}_{\mathrm{lit}}\big), \tag{6}$$

Here $\mathcal{R}_{\mathrm{ent}}$ sharpens discrete choices, $\mathcal{R}_{\mathrm{gate}}$ penalizes mean gate activation, and $\mathcal{R}_{\mathrm{lit}}$ penalizes expected literals per active clause. After warm-up, η_t increases as τ decreases. Adam optimizes the parameters between repair events; validation selects the checkpoint restored before the final discrete selection stages. Appendix B.1 specifies the schedules. Optional soft–hard alignment and hard-supervision losses are disabled in all reported Logic-KAN runs.

3.3 BOUNDED STRUCTURAL EDITING

Structural editing changes the current hard program while preserving the same bounded grammar. Its eight actions can allocate, tighten, generalize, rewrite, or remove clauses; detach clause–head memberships; switch a head between OR and XOR; or delete a head. These actions explicitly change

clause content and program topology, while gradient updates optimize their continuous parameters. Candidate construction and capacity constraints are specified in Appendix B.2.

For a binary predictor f, let

$$\mathcal{R}(f) = \sum_{i=1}^{n} w_i \, \mathbb{I}[f(\mathbf{x}_i) \neq y_i], \qquad w_i = \begin{cases} (2n_+)^{-1}, & y_i = 1, \\ (2n_-)^{-1}, & y_i = 0, \end{cases} \tag{7}$$

where n_+ and n_- count positive and negative training examples; if either class is absent, use $w_i = 1/n$. Let $\mathcal{A}(e)$ be the heads included in the local comparison, containing every head whose training outputs or head structure change under e (Appendix B.2). Let $f_{\mathcal{A}(e)}$ be their OR. The local score is

$$G(e) = \mathcal{R}(f_{\mathcal{A}(e)}) - \mathcal{R}(f'_{\mathcal{A}(e)}) - \lambda_L \Delta L - \lambda_M \Delta M - \lambda_H \Delta H, \tag{8}$$

Here $\Delta L = L_{\text{after}} - L_{\text{before}}$, with analogous definitions for ΔM and ΔH. The quantities L, M, and H count literals over distinct hard clauses, clause occurrences across heads, and nonempty heads after within-head OR deduplication and XOR pair cancellation. The cost is measured before the compiler merges OR heads and absorbs redundant clauses, preserving an incentive to remove redundant active structure. We also record each candidate's complete program error.

For a fixed candidate e, write the affected subexpression before and after the edit as $a = f_{\mathcal{A}(e)}$ and $a' = f'_{\mathcal{A}(e)}$. On every training input the OR of unaffected heads, u, is unchanged, so the full predictions are $f = a \vee u$ and $f' = a' \vee u$. Define the change in local error on training input $\mathbf{x}_i$ by

$$\Delta_i = \mathbb{I}[a(\mathbf{x}_i) \neq y_i] - \mathbb{I}[a'(\mathbf{x}_i) \neq y_i]. \tag{9}$$

If $u(\mathbf{x}_i) = 0$, the full predictions equal $a(\mathbf{x}_i)$ and $a'(\mathbf{x}_i)$; if $u(\mathbf{x}_i) = 1$, both full predictions are true. Hence the risk reductions, before complexity penalties, satisfy

$$\begin{aligned} \Delta\mathcal{R}_{\text{local}} &= \mathcal{R}(a) - \mathcal{R}(a') = \sum_i w_i \Delta_i, \\ \Delta\mathcal{R}_{\text{global}} &= \mathcal{R}(f) - \mathcal{R}(f') = \sum_i w_i \big(1 - u(\mathbf{x}_i)\big) \Delta_i. \end{aligned} \tag{10}$$

The global score therefore masks changes on inputs where another head fires. On a negative example covered by several heads, removing a false positive from the affected subexpression improves its local risk even though the full output remains wrong. This supplies a signal for correcting overlapping errors one edit at a time. Appendix A.4 proves the masking identity and bounds the difference between the local and full-program risk changes.

Local gain can also reward positive coverage already supplied by another head, so it need not reduce full-program error. An edit is eligible if the current training error is nonzero or the edited program preserves zero error. Among eligible candidates, the highest score above ϵ is accepted.

Accepted edits update both structural masks and trainable parameters; subsequent gradient steps adapt literal choices, memberships, and operators to the new structure. The bounded search algorithm, budgets, initialization, and optimizer state are reported in Appendices B.4 and B.3.

3.4 Post-training minimization and joint program selection

Local edits are effective when a useful change can be reached by modifying a small part of the current program, but some Boolean structures require a coordinated change. Logic-KAN therefore completes program selection in two stages. Empirical minimization removes literals, memberships, clauses, or heads only when training and validation risks do not increase and the program improves in fit or lexicographic structural cost.

Guarded joint program selection complements these local moves with complete candidates derived from training data, including affine-XOR, positive closure, and signed-count patterns. A candidate must fit training and validation exactly, fit the allocated grammar, and pass hard and soft installation checks; otherwise the checkpoint-selected program is retained. This stage is an inductive preference over sample-consistent programs, not a correctness certificate. Candidate construction, ordering, fallback rules, and budgets are given in Appendix B.5; additional guard conditions are summarized in Appendix C.5. Test examples, target formulas, and SAT counterexamples never participate in selection.

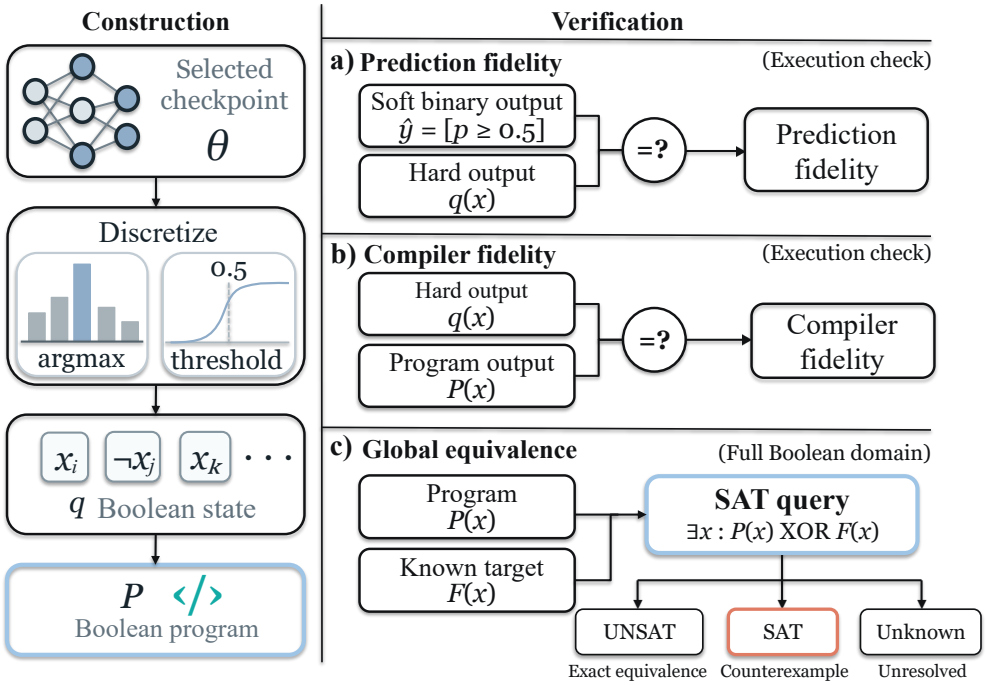


Figure 2: **Discretization, compilation, and verification.** The diagram shows the export checks applied after Logic-KAN completes empirical minimization and joint program selection. Literal argmax is followed by the six-literal projection, and operator argmax selects OR or XOR. A threshold of 0.5 converts gates and memberships to binary states. The compiler maps the final hard state to a standalone Boolean program. Prediction fidelity compares thresholded soft outputs with the hard executor, compiler fidelity compares the hard executor with the compiled program, and a SAT query checks the program against a known target over the full Boolean domain.

3.5 Compilation and Verification

Compilation translates the final hard state into conjunctions, OR/XOR heads, and a fixed outer OR, executed by a separate Boolean evaluator. It merges OR heads under the fixed outer OR, deduplicates their clauses, and uses absorption $A \vee (A \wedge B) = A$ to remove redundant OR terms. Within XOR heads it cancels duplicate clause pairs; it also removes empty heads. These transformations use neither labels nor target formulas (Figure 2).

Verification separates three sources of disagreement (Appendix A). Soft–hard comparison measures discretization effects; hard–compiled comparison checks whether export preserves the selected program; and target equivalence asks whether that executable program recovers the generating function.

For a known synthetic target F, a SAT solver checks

$$\exists \mathbf{x} \in \{0,1\}^d : P(\mathbf{x}) \oplus F(\mathbf{x}) = 1. \tag{11}$$

Unsatisfiability proves exact equivalence; satisfiability returns a checked counterexample. Target formulas generate the supervision labels. Parameter learning, candidate search, and checkpoint selection use the resulting examples without access to formula structure or SAT feedback. Equivalence queries run after all data-based selection stages and final compilation.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Tasks and data. We evaluate 69 synthetic tasks: nine classical Boolean tasks (parity, conjunction, majority, and multiplexers), five fixed constructed tasks (nested logic and planted DNF), and 55 random teacher instances. Inputs contain 6–100 Boolean features. Random DNF teachers vary in dimension and clause count, with three literals per clause. Mixed teachers combine shared clauses through OR/XOR heads.

A teacher seed identifies the target function; a run seed controls data sampling and optimization. Each task uses five run seeds, 0–4, giving 345 development-catalogue runs. The external comparison covers DNF and Mixed targets in two teacher batches, 100–109 and 130–139. Each family and batch contains ten teachers with five runs each. The four conditions therefore contain 200 runs per method: 1000 runs across the five external methods and 200 corresponding Logic-KAN runs included in the main catalogue.

Inputs are sampled uniformly without replacement into disjoint training, validation, and test sets. Tasks with at most 13 inputs use 75/12.5/12.5% of the full domain; larger tasks use 4096/1024/4096 examples. All methods in the external comparison use the same frozen arrays for each teacher and run seed. Appendix C.1, Table 6, lists the tasks and split sizes.

We additionally ran a versioned post-development matrix of 116 jobs and 580 complete runs. Block B1 is a blind evaluation of the complete model on 30 teacher functions with seeds 200–209 (150 runs). Block B2 contains 400 paired mechanism-control runs on ten anchor teachers, and Block B3 contains 30 targeted candidate-attribution runs. The 580-run count includes controls; the complete model's blind result is reported separately from the ablations. For the B2 paired comparisons, the complete-model reference is the 50-run subset of B1 on the ten DNF/XOR anchor teachers; it is reused as a paired baseline rather than counted as another 50 unique runs.

Training and experimental records. All Logic-KAN runs use 96 clause slots, eight heads, and at most six literals per clause, with the learning and selection procedure in Section 3. Complete training records are retained for Parity16, Majority7, and Conjunction, each with five run seeds, giving 15 runs. The remaining 330 main runs retain final predictions, programs, selection records, and target-equivalence certificates. Hyperparameters and execution order appear in Appendix B; method configurations and execution environments appear in Appendix C.3. Artifact coverage is documented in Appendix C.8.

Evaluation metrics. Accuracy and balanced accuracy are computed from the final hard predictions. Exact recovery requires the exported program to agree with the target on every Boolean input. We use the counterexample query in Equation (11). An unsatisfiable query certifies equivalence; a satisfiable query identifies an input on which the functions disagree. All retained queries have a resolved outcome.

Catalogue accuracy averages the per-task mean accuracies. External comparison accuracy is reported as the mean and sample standard deviation across teacher means. Exact recovery counts all runs within the stated block. For the post-development matrix we also report teacher-level bootstrap intervals for paired recovery differences. Soft–hard disagreement measures the fraction of test inputs on which thresholded continuous predictions differ from hard predictions. Appendix C.4 specifies the auxiliary metrics.

4.2 RECOVERING FUNCTIONS BEYOND THE OBSERVED EXAMPLES

Logic-KAN attains a task-macro accuracy of 99.705% and exact recovery in 333/345 runs (96.522%). All five seeds recover the target in 61/69 tasks. Recovery is complete for all 25 fixed-constructed runs and for 270/275 random-teacher runs. Classical tasks account for the remaining 38/45 successful runs, including perfect recovery of the four parity tasks, Majority7, and Mux8.

Table 1 separates these results by target family. Logic-KAN is not restricted to explicit DNF teachers: it recovers all 20 XOR/parity runs, including all four parity tasks, and all five nested-logic runs. It also recovers 155/160 Random-DNF runs and all 115 Random-Mixed runs. The Mixed teachers

Table 1: Recovery by target family. Accuracy is the mean across runs within each family; Exact counts programs certified equivalent to the target.

Target family	Tasks	Runs	Accuracy (%)	Exact
XOR/parity	4	20	100.000	20/20
Conjunction/count	2	10	97.500	8/10
Multiplexer	3	15	95.688	10/15
Nested logic	1	5	100.000	5/5
Planted DNF	4	20	100.000	20/20
Random DNF	32	160	99.924	155/160
Random Mixed	23	115	100.000	115/115

Table 2: Post-development blind evaluation on 30 unseen teachers. Each teacher has five run seeds. The interval is a teacher-level bootstrap 95% confidence interval for the final exact-recovery rate.

Family	Teachers	Runs	Checkpoint	Minimized	Final (95% CI)
Random DNF	10	50	43/50	49/50	49/50 [.940,1.000]
Random XOR	10	50	48/50	48/50	48/50 [.880,1.000]
Random Mixed	10	50	49/50	50/50	50/50 [1.000,1.000]

use OR/XOR heads and reduce to seven-clause DNF (Appendix A.5). Recovery rates are lower for conjunction/count and multiplexer tasks.

The complete grouped catalogue and all 69 per-task rows are reported in Appendix C.1, Tables 5 and 6.

4.3 Post-development validation and mechanism evidence

The post-development blind block evaluates the complete Logic-KAN specification on teacher functions whose seeds were not used in the development catalogue. Table 2 reports the exact recovery at the checkpoint, after empirical minimization, and after final selection. The final blind result is 147/150 (98.0%); all 49 SAT-exact Random-DNF exports in this block also match their teacher signed-clause sets.

The paired mechanism block holds teachers, sampled arrays, and run seeds fixed while changing one control configuration. The complete pipeline recovers 47/50 anchor runs. Freezing the ternary literal edges reduces recovery to 0/50, global rather than local scoring to 20/50, OR-only heads to 23/50 (0/25 on Random XOR), and unshared clauses to 32/50 (7/25 on Random XOR). The gradient-only control recovers 28/50 and accounts for ten of the twelve perfect-test but SAT-non-equivalent formal runs. These controls support the role of the Boolean edge vocabulary, local scoring, XOR heads, and clause sharing, while the gradient-only comparison removes several mechanisms at once and is therefore not a single-component attribution.

The remaining controls delimit the claim rather than strengthen it: removing joint candidates ties the complete pipeline at 47/50 on these random anchors, the fixed-capacity control reaches 41/50 with an interval crossing zero, and the growth-only control reaches 50/50. The joint candidates are therefore structure-specific inductive biases: removing affine-XOR reduces Parity16 from 5/5 to 0/5, removing signed-count reduces Majority7 from 5/5 to 0/5, and removing positive closure reduces Conjunction from 3/5 to 0/5. Their complete attribution table is in Appendix C.6.

Across all 580 formal runs, 413 have 100% hard test accuracy, but only 401 are SAT exact; the remaining twelve are hidden mismatches (ten gradient-only, one complete Conjunction run, and one no-closure Conjunction run). The blind complete-model block has no such hidden mismatch. This separation is a direct empirical reason to report SAT equivalence alongside finite-sample accuracy.

4.4 Similar predictive accuracy, different functional recovery

Comparison protocol. We compare Logic-KAN with Neural DNF, using the semi-symbolic layers of pix2rule (Cingillioglu & Russo, 2021), RRL (Wang et al., 2021), DiffLogic (Petersen et al., 2022),

Table 3: **Comparison with external rule-learning methods.** Accuracy is the mean and sample standard deviation across teacher means; Exact counts programs certified equivalent to the target. Bold marks the best or tied-best unrounded result within each condition.

	Teacher seeds 100–109			
	DNF32-C12-L3		Mixed32-C3-L3-H3	
Method	Acc. (%)	Exact	Acc. (%)	Exact
Logic-KAN	99.757 ± 0.338	**45/50**	$\mathbf{100.000 \pm 0.000}$	**50/50**
RRL	$\mathbf{99.934 \pm 0.031}$	0/50	99.984 ± 0.028	36/50
Neural DNF	95.069 ± 2.078	1/50	98.940 ± 1.121	31/50
BRCG	93.662 ± 1.443	0/50	$\mathbf{100.000 \pm 0.000}$	**50/50**
DiffLogic	84.974 ± 1.036	0/50	84.046 ± 0.659	0/50
BoolXAI	67.603 ± 2.144	0/50	68.921 ± 1.898	0/50

	Teacher seeds 130–139			
	DNF32-C12-L3		Mixed32-C3-L3-H3	
Method	Acc. (%)	Exact	Acc. (%)	Exact
Logic-KAN	$\mathbf{100.000 \pm 0.000}$	**50/50**	$\mathbf{100.000 \pm 0.000}$	**50/50**
RRL	99.947 ± 0.027	0/50	99.998 ± 0.006	44/50
Neural DNF	95.336 ± 1.809	0/50	99.057 ± 1.348	36/50
BRCG	92.729 ± 1.423	0/50	$\mathbf{100.000 \pm 0.000}$	**50/50**
DiffLogic	84.615 ± 0.508	0/50	84.577 ± 0.855	0/50
BoolXAI	66.219 ± 2.748	0/50	71.706 ± 1.972	0/50

BRCG (Dash et al., 2018), and BoolXAI (Kadioglu et al., 2025). The comparison uses DNF32-C12-L3 and Mixed32-C3-L3-H3 under the data protocol above. Capacity, learning rates, and search budgets are method-specific; Appendix C.3 records the implementations and configurations. Table 3 reports test accuracy and exact recovery for both teacher batches.

Functional and structural recovery. Table 3 shows that accuracy rankings differ from exact-recovery rankings; Appendix C.4, Figures 3 and 4 show the same separation across teacher batches. On DNF teachers 100–109, RRL has slightly higher accuracy than Logic-KAN ($99.934 \pm 0.031\%$ versus $99.757 \pm 0.338\%$) but recovers 0/50 targets, whereas Logic-KAN recovers 45/50. On teachers 130–139, Logic-KAN recovers 50/50 while RRL reaches $99.947 \pm 0.027\%$ accuracy but 0/50 exact recovery. Across both DNF batches, Logic-KAN recovers 95/100 targets, compared with 1/100 for Neural DNF and 0/100 for every other baseline. Among the five external methods, 301 runs are perfect on the 4096 test inputs, yet 53 exported programs are not target-equivalent; all 195 perfect-test Logic-KAN exports are exact. Auxiliary metrics are reported in Appendix C.4, Table 9. Recovery also varies by target family: Logic-KAN and BRCG recover all 100 Mixed targets, while the four-condition exact counts are 195/200, 100/200, 80/200, 68/200, and 0/200 for Logic-KAN, BRCG, RRL, Neural DNF, and each of DiffLogic and BoolXAI, respectively. The Mixed teacher is equivalent to a seven-clause DNF despite its OR/XOR construction (Appendix A.5), so these results test functional recovery without requiring its original head layout. For structural recovery, all 155 SAT-exact development Random-DNF exports and all 49 SAT-exact blind exports match the teacher signed-clause set; the full audit is in Appendix C.2, Table 7.

5 Conclusion

Logic-KAN makes Boolean rules directly editable during learning, combining gradient optimization with bounded structural changes in a shared grammar. The local-risk analysis explains how the outer OR masks rule corrections, while compilation turns the selected structure into an executable program. Experiments demonstrate recovery across diverse Boolean tasks, with higher exact recovery on DNF targets than the evaluated baselines and complete recovery on Mixed targets. The separation between test accuracy and exact recovery highlights the value of treating learned rules as computational objects whose structure can be optimized and whose semantics can be verified.

AI USE STATEMENT

In this work, we used generative AI tools for manuscript language polishing, formula typesetting, and figure caption refinement. We did not use generative AI tools for the development of the core methodology, theoretical or mathematical claims, experimental design or implementation, or result analysis and interpretation; other research tasks requiring disclosure, such as synthetic data generation and qualitative data analysis, are not applicable to this work. Additionally, we used generative AI tools for preliminary literature-search assistance. We reviewed all AI-assisted work and manually cross-checked AI-polished text against our original arguments. All experiment-related descriptions were independently verified by two authors. We take full responsibility for the final content of this work, including any text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This study evaluates LogicKAN on simulated, synthetic Boolean benchmark tasks. No new data are collected from human participants, and no experiments involving human subjects or deployed decision-making systems are conducted. The datasets are used for model training, validation, and evaluation within the experimental protocols described in this paper.

REPRODUCIBILITY STATEMENT

Code and Implementation. Anonymous source code, generated datasets, configuration files, and scripts for running training and evaluation are provided in the supplementary material archive. Logic-KAN is implemented in PyTorch. The recorded environment for runs with complete training artifacts uses an NVIDIA A100-SXM4 GPU with 40 GB memory, Python 3.12.13, PyTorch 2.11.0+cu128, CUDA 12.8, and NumPy 2.4.6. Computation uses float32 without automatic mixed precision. Software dependencies are listed in `requirements.txt`. Training procedures and method-specific configurations are described in Appendices B and C.3.

Datasets. We evaluate Logic-KAN on 69 synthetic Boolean tasks covering classical functions, constructed rules, and randomly generated targets. Target definitions, generation procedures, and split sizes are described in Appendix C.1. Tasks with at most 13 inputs use 75/12.5/12.5% of the Boolean domain for training, validation, and testing; larger tasks use 4,096/1,024/4,096 examples. Inputs are sampled uniformly without replacement, and the three splits are disjoint. All methods in the external comparison use the same frozen data arrays for each teacher and run seed.

Hyperparameters and Experimental Protocols. Table 4 summarizes the Logic-KAN hyperparameters and structural search budgets. Training uses Adam with a learning rate of 0.03 and a batch size of 256 for 400 epochs. Each task uses five run seeds, 0–4, which control data sampling and optimization. Validation data guide checkpoint and final program selection. Section 4.1 defines the evaluation metrics and aggregation procedures, while Appendix C.4 details the external comparison and verification semantics. Target formulas generate the labels; training and program selection use these examples without access to formula structure or SAT feedback. Appendix C.8 documents the coverage of retained experimental records.

REFERENCES

Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research*, 18(234): 1–78, 2018.

Kexin Gu Baugh, Vincent Perreault, Matthew Baugh, Luke Dickens, Katsumi Inoue, and Alessandra Russo. Disentangling neural disjunctive normal form models. In *Proceedings of the 19th International Conference on Neurosymbolic Learning and Reasoning*, volume 284 of *Proceedings of Machine Learning Research*, pp. 463–493. PMLR, 2025.

Nuri Cingillioglu and Alessandra Russo. pix2rule: End-to-end neuro-symbolic rule learning. In *Proceedings of the 15th International Workshop on Neural-Symbolic Learning and Reasoning*, volume 2986 of *CEUR Workshop Proceedings*, pp. 15–56. CEUR-WS.org, 2021.

Sanjeeb Dash, Oktay Günlük, and Dennis Wei. Boolean decision rules via column generation. In *Advances in Neural Information Processing Systems*, volume 31, pp. 4660–4670. Curran Associates, Inc., 2018.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018. doi: 10.1613/jair.5714.

Katarzyna Fojcik, Renaldas Zioma, and Jogundas Armaitis. LILogic Net: Compact logic gate networks with learnable connectivity for efficient hardware deployment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 3498–3507, 2026.

Serdar Kadioglu, Elton Yechao Zhu, Gili Rosenberg, John Kyle Brubaker, Martin J. A. Schuetz, Grant Salton, Zhihuai Zhu, and Helmut G. Katzgraber. BoolXAI: Explainable AI using expressive Boolean formulas. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(28):28900–28906, 2025. doi: 10.1609/aaai.v39i28.35157.

Connor Lawless, Sanjeeb Dash, Oktay Gunluk, and Dennis Wei. Interpretable and fair Boolean rule sets via column generation. *Journal of Machine Learning Research*, 24(229):1–50, 2023.

Zhaoyu Li, Jinpei Guo, Yuhe Jiang, and Xujie Si. Learning reliable logical rules with SATNet. In *Advances in Neural Information Processing Systems*, volume 36, pp. 14837–14847, 2023. doi: 10.52202/075280-0654.

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljacic, Thomas Hou, and Max Tegmark. KAN: Kolmogorov–Arnold networks. In *International Conference on Learning Representations*, 2025.

Dmitry Malioutov and Kush Varshney. Exact rule learning via Boolean compressed sensing. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pp. 765–773. PMLR, 2013. Part 3.

Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Deep differentiable logic gate networks. In *Advances in Neural Information Processing Systems*, volume 35, pp. 2006–2018, 2022. doi: 10.52202/068431-0146.

Litao Qiao, Weijia Wang, and Bill Lin. Learning accurate and interpretable decision rule sets from neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5):4303–4311, 2021. doi: 10.1609/aaai.v35i5.16555.

Jan Niclas Reimann, Andreas Schwung, and Steven X. Ding. Neural logic rule layers. *Information Sciences*, 596:185–201, 2022. doi: 10.1016/j.ins.2022.03.021.

Lukas Rüttgers, Till Aczel, Andreas Plesner, and Roger Wattenhofer. Light differentiable logic gate networks. In *International Conference on Learning Representations*, 2026.

Taisuke Sato and Katsumi Inoue. Differentiable learning of matricized DNFs and its application to Boolean networks. *Machine Learning*, 112(8):2821–2843, 2023. doi: 10.1007/s10994-023-06346-5.

Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6545–6554. PMLR, 2019.

Zhuo Wang, Wei Zhang, Ning Liu, and Jianyong Wang. Scalable rule-based representation learning for interpretable classification. In *Advances in Neural Information Processing Systems*, volume 34, pp. 30479–30491, 2021.

Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems*, volume 30, pp. 2319–2328. Curran Associates, Inc., 2017.

Shakir Yousefi, Andreas Plesner, Till Aczel, and Roger Wattenhofer. Mind the gap: Removing the discretization gap in differentiable logic gate networks. In *Advances in Neural Information Processing Systems*, volume 38, pp. 122454–122484. Curran Associates, Inc., 2025. doi: 10.52202/085713-4085.

A ASSUMPTIONS, DERIVATIONS, AND PROOFS

A.1 OBJECTS AND ASSUMPTIONS

Inputs belong to $\{0,1\}^d$ and labels to $\{0,1\}$. A clause has at most one signed literal per feature; omission contributes the constant true value. Let q denote a valid hard state, with ternary literal indices, binary slot and head masks, binary memberships, and one OR or XOR choice per head. "Valid" means that these arrays have the shapes and value ranges checked by `compile_program`. Structural masks are combined with the learned gate decisions before this state is compiled. An empty conjunction is true, an empty head is false, and the OR of no retained heads is false.

Literal and operator states use the first maximum returned by `argmax` when logits tie. If more than K features select a positive or negative literal, the hard projection retains the K features with the largest margin between their selected signed-literal logit and the omission logit. Equal margins preserve ascending feature order through a stable descending sort; all remaining features are mapped to omission. Clause, membership, and head gates use the inclusive threshold $\sigma(z) \geq 0.5$ and are intersected with their structural masks. These rules are shared by the hard executor and compiler.

The exact identities below concern Boolean execution or the algebraic soft operations before numerical clipping. They require neither independence of shared clauses nor a distributional assumption about the target. The statements about empirical risk hold for fixed, nonnegative weights on a nonempty training set. In the implemented balanced risk, every weight is positive and their sum is one. The hard literal projection precedes these Boolean operations; it is not an algebraic identity of the unconstrained soft predictor.

A.2 BOOLEAN SEMANTICS OF THE RELAXED OPERATORS

Proposition A.1 (Boolean reduction). If all categorical distributions are replaced by one-hot choices that respect the hard literal cap, and all gates by binary values, the unclipped algebraic predictor evaluates the corresponding hard program.

Proof. A one-hot literal choice makes ϕ_{sj} equal to x_j, $1 - x_j$, or 1. Its product over features is therefore the conjunction specified by that slot. A zero clause gate or membership contributes a false input to a head. For Boolean inputs $b_1, \ldots, b_k$,

$$1 - \prod_{i=1}^{k} (1 - b_i) = \bigvee_{i=1}^{k} b_i, \qquad \frac{1 - \prod_{i=1}^{k} (1 - 2b_i)}{2} = \bigoplus_{i=1}^{k} b_i. \tag{12}$$

For the first identity, the product is one exactly when every input is zero. For the second, each true input contributes a factor of -1, so the product is $(-1)^{\sum_i b_i}$. A one-hot operator selects the required identity. Multiplying by the binary head gate removes disabled heads, and applying the first identity once more gives the outer OR. The empty-product convention gives false for both empty heads. This proves the claim.

These are algebraic identities, including when inputs to different heads share clauses. The soft values need not be interpreted as probabilities of independent logical events. Finite softmax logits do not give one-hot distributions, and the implementation clips intermediate values and final probabilities. In particular, an algebraic output of one is returned as at most $1 - 10^{-6}$ after clipping. Proposition A.1 identifies the discrete semantics; it does not assert exact equality of finite-parameter soft and hard predictions.

Corollary A.2 (Representation within the allocated grammar). A signed DNF with C distinct clauses, each containing at most K literals, has a hard representation when $S \geq C$ and at least one head is available. Allocate one slot per clause and connect them to a single OR head. A parity of C signed conjunctions has the analogous representation with a single XOR head. These constructions follow directly from Proposition A.1 and use the full ternary literal arrays. The final configuration enforces $K = 6$ on both hard literal selection and proposals; the representation statement is restricted to clauses within that bound. The construction makes no assertion about finding the state by optimization.

A.3 CORRECTNESS OF HARD-STATE COMPILATION

Proposition A.3 (Compilation preserves the hard function). For every valid hard state q and binary input $\mathbf{x}$, the Boolean program P_q emitted by the compiler satisfies

$$P_q(\mathbf{x}) = f_q(\mathbf{x}). \tag{13}$$

Proof. The compiler first skips disabled heads. Within an enabled head it visits exactly the slots whose hard slot mask and membership are both true, the same conjunction of masks used by the hard executor. Scanning each slot's literal indices emits the positive or negated input for indices 0 or 1 and omits index 2. The resulting clause therefore agrees with the hard clause on every binary input, including the empty conjunction.

It remains to check simplification. For an OR head, repeating the same clause does not change its value because $b \vee b = b$. The compiler retains its first occurrence. For an XOR head, two identical clauses cancel because $b \oplus b = 0$; retaining a clause exactly when its multiplicity is odd preserves parity. This also holds for duplicate empty conjunctions, whose value is true. After these operations, a head with no clauses has value false and may be removed from the outer OR. Every remaining head has the original value. Since the outer operation is OR, associativity permits merging all OR heads. If the literals of clause A are a subset of those of clause B, then B implies A, so $A \vee B = A$ and B can be removed. Deduplication and absorption of the merged OR terms therefore preserve the full function. The independent evaluator computes precisely the resulting conjunctions and head operations.

Sharing across heads does not change this proof: it reuses a clause function without changing its value. The compiler performs within-head duplicate handling and OR merging and absorption, not general Boolean minimization. The literal count in a final exported program and the pre-absorption cost used in edit scoring are representation measures, not the minimum formula size of the represented function.

A.4 EFFECT OF AN EDIT IN A FIXED OR CONTEXT

Fix a candidate edit and the affected set used by its scorer. The set includes every head whose training-output column changes and the relevant changed memberships, operators, and activation flags. For add, refine, replace, and delete, it includes all heads attached to the edited slot before or after the edit, including a new head. All other heads have unchanged outputs on every training example. Let a, a' be the affected OR before and after the edit and u the OR of unaffected heads. For any binary label y, define $\ell_y(b) = \mathbb{I}[b \neq y]$.

Proposition A.4 (Context masking and its bound). The local and global risk changes in the main text satisfy the stated masking identity. Moreover,

$$\begin{aligned} \Delta\mathcal{R}_{\text{local}} - \Delta\mathcal{R}_{\text{global}} &= \sum_i w_i u(\mathbf{x}_i)\big[\ell_{y_i}(a(\mathbf{x}_i)) - \ell_{y_i}(a'(\mathbf{x}_i))\big], \\ |\Delta\mathcal{R}_{\text{local}} - \Delta\mathcal{R}_{\text{global}}| &\leq \sum_i w_i u(\mathbf{x}_i)\mathbb{I}[a(\mathbf{x}_i) \neq a'(\mathbf{x}_i)]. \end{aligned} \tag{14}$$

Proof. At an input with $u = 0$, the full outputs are a and a', so local and global error changes agree. At an input with $u = 1$, both full outputs are one, and their error difference is zero. Thus the pointwise global difference is $(1 - u)(\ell_y(a) - \ell_y(a'))$. Multiplication by w_i and summation gives the main-text identity. Subtracting it from the local sum yields the first line above. For Boolean a, a', the absolute error difference is zero when they agree and one when they differ. Applying the triangle inequality and nonnegativity of the weights proves the bound.

For a negative example with $a = 1$, $a' = 0$, and $u = 1$, the local error falls while the full output stays wrong. Conversely, adding positive coverage where $u = 1$ earns local credit without changing the full prediction. Both cases follow from the same identity. The local and global penalized scores subtract the same complexity change, so their difference is still the first line of Equation (14). Positive local gain alone does not imply a decrease in global risk. This is a property of the scoring rule, not an empirical claim about edit trajectories in the retained catalogue.

Proposition A.5 (Scope of the zero-error guard). If a repair event begins at zero training error, every accepted intermediate state of that event also has zero training error.

Proof. For the first candidate, the implementation makes zero risk after the edit an eligibility condition because risk before the edit is zero. Every accepted candidate must satisfy that condition, regardless of its local score. The accepted state becomes the current state for the next round, where the same reasoning applies. Induction over accepted edits proves the statement. Rejected rounds leave the state unchanged. Positive sample weights make zero risk equivalent to classifying every training example correctly.

Gradient updates between events are outside this argument. The guard also does not constrain an event's first edit when its incoming training error is positive. Neither training-set statement entails equivalence to the target on unseen inputs.

A.5 The mixed-labelled generator simplifies to DNF

The saved mixed-labelled tasks use seven signed conjunctions a, b, c, d, e, f, g, arranged as $\mathrm{OR}(a, b, c)$, $\mathrm{XOR}(c, d, e)$, and $\mathrm{OR}(e, f, g)$, with a fixed outer OR. This is the concrete overlap pattern produced by the generator with three heads and three clauses per head.

Proposition A.6 (Generator identity). For every assignment to these seven clause values,

$$(a \vee b \vee c) \vee (c \oplus d \oplus e) \vee (e \vee f \vee g) = a \vee b \vee c \vee d \vee e \vee f \vee g. \tag{15}$$

Proof. If c or e is true, an outer OR head already makes both sides true. Otherwise $c = e = 0$, so the middle XOR equals d and the remaining outer expression is $a \vee b \vee d \vee f \vee g$. This equals the right-hand side under the same condition. The cases exhaust the domain.

The argument allows arbitrary dependence among the seven conjunctions. The identity is also verified by enumeration of all $2^7 = 128$ assignments to the seven clause values. The results are therefore reported as DNF-equivalent Mixed tasks, separately from the twelve-clause DNF family.

B Training and implementation details

Table 4 specifies the final Logic-KAN configuration. An initial clause count of zero selects the default basis of $\min(8, d, S)$ positive singleton clauses and one head. The nine scheduled repair events occur after epochs 40, 80, 120, 160, 200, 240, 280, 320, and 360. Validation is measured before each event; the edit affects the next epoch. With $u_t = (t-1)/(T-1)$, the condition $u_t < 0.90$ still permits the epoch-360 event. A candidate budget is an upper bound, not a claim that every event exhausts it.

Table 4: Final Logic-KAN training and structural search settings. Complete end-to-end training artifacts are retained for 15 runs; the other 330 final entries retain the inputs and outputs needed to audit the reported results.

Quantity	Value	Quantity	Value
Maximum slots	96	Maximum heads	8
Initial clauses	$\min(8, d, S)$	Initial heads	1
Epochs	400	Batch size	256
Adam learning rate	0.03	Gradient norm cap	5
Initial temperature	1.0	End temperature	0.3
Warm-up fraction	0.35	Edit stop fraction	0.90
Repair interval	40	Candidate budget/event	256
Maximum edits/event	4	Hard literal cap	6
Minimum gain	10^{-8}	Optimizer after edits	reset
λ_{ent}	0.005	λ_{gate}	0.001
λ_{lit}	0.0001	λ_L, λ_M	0.0001
λ_H	0.001	SAT timeout	10 s
Minimization steps	64	Joint candidates	64
Alignment weight	0	Hard-supervision weight	0

The complete training and export sequence is:

1. activate the initial clause basis and one logic head;
2. update the temperature and regularization schedules, then optimize continuous parameters for one epoch;
3. evaluate the pre-edit state on validation data and update the retained checkpoint using the criterion below;
4. at each scheduled event, run Algorithm 1 with at most 256 candidate evaluations over at most four rounds;
5. repeat steps 2–4 through epoch 400, then restore the selected checkpoint and its temperature;
6. perform at most 64 steps of empirical minimization, followed by guarded joint program selection with at most 64 retained candidates;
7. compile the final hard state and apply the independent execution and target-equivalence checks.

B.1 Regularization and schedules

When either active index set is empty, all three regularizers are defined to be zero; otherwise, they are given by: Let $\mathcal{S}$ and $\mathcal{T}$ be the structurally active slot and head index sets. The gate vector $\mathcal{G}$ concatenates all g_s, m_{hs}, and b_h for $s \in \mathcal{S}$, $h \in \mathcal{T}$, so $|\mathcal{G}| = |\mathcal{S}| + |\mathcal{T}||\mathcal{S}| + |\mathcal{T}|$. With natural logarithms and $\delta = 10^{-9}$, the implementation uses

$$\mathcal{H}_\delta(p) = -\sum_k p_k \log \max(p_k, \delta), \tag{16}$$

$$h_\delta(z) = -z \log \max(z, \delta) - (1 - z) \log \max(1 - z, \delta), \tag{17}$$

$$\begin{aligned}\mathcal{R}_{\text{ent}} = {} & \frac{1}{|\mathcal{S}|d} \sum_{s \in \mathcal{S}} \sum_{j=1}^{d} \mathcal{H}_\delta(\boldsymbol{\alpha}_{sj}) \\ & + \frac{1}{|\mathcal{T}|} \sum_{h \in \mathcal{T}} \mathcal{H}_\delta(\boldsymbol{\pi}_h) + \frac{1}{|\mathcal{G}|} \sum_{z \in \mathcal{G}} h_\delta(z),\end{aligned} \tag{18}$$

$$\mathcal{R}_{\text{gate}} = \frac{1}{|\mathcal{G}|} \sum_{z \in \mathcal{G}} z, \qquad \mathcal{R}_{\text{lit}} = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \sum_{j=1}^{d} (1 - \alpha^0_{sj}). \tag{19}$$

Each entropy family is averaged separately before summation. Gate sparsity instead uses a single mean over the concatenated vector; literal cost sums over features and averages over active slots. These terms use structural masks, without thresholding the learned gates. All three return zero if either active index set is empty.

For epoch $t \in \{1, \ldots, T\}$, the warm-up fraction $w = 0.35$ determines both the regularization scale and temperature:

$$u_t = \frac{t - 1}{\max(1, T - 1)}, \qquad \eta_t = \max\left(0, \frac{u_t - w}{1 - w}\right), \qquad \tau_t = \tau_0 \left(\frac{\tau_f}{\tau_0}\right)^{\eta_t}. \tag{20}$$

Here $\tau_0 = 1$ and $\tau_f = 0.3$. Thus the temperature stays at τ_0 and the regularization scale at zero during warm-up. The value 0.3 is the schedule endpoint; export restores the temperature of the selected checkpoint, which may precede the final epoch. The optional fidelity losses have zero weight in the retained results.

BCE is averaged over each mini-batch. The soft logic formulas omit numerical stabilization: literal values are clipped below at 10^{-9} before taking logs, and the exponentiated clause product is clipped to $[0, 1]$. Both u_{hs} and the gated head outputs entering the outer OR are clipped to $[0, 1 - 10^{-6}]$; the final prediction is clipped to $[10^{-6}, 1 - 10^{-6}]$. With no active slots or heads the soft predictor returns 10^{-6}.

For zero-based round j, with n candidates already scored, the allowance is $b_j = \max(1, \lfloor (B - n)/(J - j) \rfloor)$, and the pool contains at most b_j candidates. Rejected rounds still count toward J,

and unused allowance carries over. Ties in G are resolved by taking the first eligible candidate in pool order. An event that accepts an edit resets Adam before the next training epoch; otherwise the optimizer is retained.

The repair pool uses a deterministic NumPy seed sequence $[\text{seed}, t, j, 1729]$ for zero-based proposal round j. Action types receive candidates in round-robin order after shuffling within each type. Allowance is based on scored candidates, including rejected ones; accepted edits cause the next pool to be generated from the updated hard state.

B.2 Candidate generation

Proposal generation distinguishes structurally allocated components from those retained by hard gate thresholds. A new clause uses the first structurally free slot. It connects to a retained head, or to the first structurally free head with either OR or XOR. Clause rewriting and deletion draw from slots used by at least one retained head. Deletion requires more than the minimum one allocated slot. A head can be deleted only when more than one retained head exists.

For each used slot, refinement samples a covered full-program false positive and a covered positive example. Each unused feature on which this pair differs supplies a literal matching the positive anchor. Refinement is allowed only below the six-literal cap. Generalization removes one literal from a clause of length greater than one. Detachment removes one selected membership, and an operator flip switches a retained head to its other operator.

Add and replace proposals sample positive training anchors. At most $4b_j$ anchor draws are made per round; each samples an order uniformly from 1 to $\min(K, d)$ and that many distinct features. Replacement rejects unchanged terms. Each action pool is deduplicated and bounded by b_j. Shuffling within types and round-robin allocation across all eight types produces at most b_j scored candidates.

For add, refine, replace, and delete, the local comparison includes every head attached to the edited slot before or after the edit, plus a newly created head. For generalize, detach, operator flip, and head deletion, it includes heads with changed training-output columns, memberships, operators, or activation. Heads excluded from the comparison have unchanged predictions on the scoring data. Scoring uses the full training set and the balanced risk in Equation (7), with uniform weights if only one class occurs.

B.3 Initialization and continuous state after an edit

Add, refine, generalize, and replace reset the edited slot's literal logits. In $(+, -, 0)$ order, omission receives $(0, 0, \tau \log(\max(2, 20d)))$, a positive literal receives $(2.8\tau, 0, 0)$, and a negative literal receives $(0, 2.8\tau, 0)$. Refine, generalize, and replace preserve the slot gate and its memberships.

Add sets the clause logit to 3, the membership column to -3, and the selected membership to 2. A new head has gate logit 2, its membership row initialized to -3, and operator logits 2τ and -2τ for the selected and alternative operators. Bootstrap uses positive singleton features chosen by a random permutation and the same clause initialization; its tied zero operator logits select OR by argmax.

Delete clears a slot's structural mask and sets its membership column to -3. Detach sets the chosen membership logit to -3. An operator flip sets the selected and alternative operator logits to 2τ and -2τ. Head deletion clears its structural mask and membership row. Adam is reset only after an event with at least one accepted edit.

B.4 Repair pseudocode

Here f_q is the full hard predictor of state q, and q_e is the state after candidate e.

B.5 Post-training selection details

Empirical minimization uses training and validation class-balanced risks. Both must weakly improve (tolerance 10^{-15}); a candidate must strictly improve a risk or reduce either the cost tuple (L, M, H) or the allocated-capacity tuple (slots, heads, selected memberships) lexicographically. Candidates are ranked by validation improvement, training improvement, total cost reduction, then the component-

Algorithm 1 Bounded structural-edit search with the eight-action proposal set

Require: Parameters θ, structural masks $\mathcal{M}$, training data D, budget B, rounds J, threshold ϵ
$q \leftarrow \mathrm{HardState}(\theta, \mathcal{M})$; $n \leftarrow 0$
for $j = 0, \ldots, J-1$ while $n < B$ **do**
  $C \leftarrow \mathrm{Propose}(q, \mathcal{M}, D, \text{remaining allowance})$
  if $C = \varnothing$ **then break**
  Score C by Eq. (8); $n \leftarrow n + |C|$
  $E \leftarrow \{e \in C : \mathcal{R}(f_q) > 0 \text{ or } \mathcal{R}(f_{q_e}) = 0\}$
  if $E \neq \varnothing$ **then**
    $e^* \leftarrow \arg\max_{e \in E} G(e)$
    if $G(e^*) > \epsilon$: update $(\theta, \mathcal{M})$ with e^*; refresh q
  end if
end for

wise cost and capacity reductions. At most 64 accepted steps are performed, with no claim of global minimization.

For joint program selection, affine candidates solve $Xw + b = y$ over $\mathrm{GF}(2)$ and are discarded when inconsistent or underdetermined. Selected features become singleton XOR clauses; an intercept is the existing empty true conjunction. A positive-closure candidate contains all literals constant across training positives and must exclude every training negative.

Signed-count candidates rank features by the absolute class-conditional mean difference and orient their signs by that difference. For each ranked prefix, let c_i count satisfied signed literals. A threshold is retained only if $1 + \max_{y_i=0} c_i = \min_{y_i=1} c_i = k$, so the observed boundary determines a unique integer threshold. The rule is expanded into all k-literal conjunctions from the prefix only when their count fits the 96 slots and $k \leq 6$. At most d prefixes and one affine solve are considered, retaining at most 64 distinct training-exact programs.

All eligible candidates must also fit validation exactly. If the checkpoint-selected program already fits both splits, only a positive-closure candidate whose literal set strictly contains that of its single conjunction may replace it. Installation sets selected logits to 20 and alternatives to -20 in the existing circuit, then checks hard and thresholded soft execution on both splits. Failed installation restores the previous state and raises an error. Test data and target verification never select among candidates.

C Additional experimental details

C.1 Catalogue and statistical units

Every task in the versioned catalogue has five runs. Teacher seeds 0–2, 100–109, and 130–139 identify different generated functions, and are kept as separate groups. For a group of T teachers, a metric m is summarized by

$$\bar{m}_t = \frac{1}{5}\sum_{r=0}^{4} m_{t,r}, \qquad \bar{m} = \frac{1}{T}\sum_{t=1}^{T} \bar{m}_t, \qquad s_{\text{teacher}} = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}(\bar{m}_t - \bar{m})^2}. \tag{21}$$

For a single fixed task, the sample standard deviation is instead computed over its five runs. These spreads are not confidence intervals, and no statistical significance test is claimed. The overall accuracy is the unweighted mean of the 69 task means. The suffix of a task ID, such as `s105`, is the teacher seed. The five run seeds change both optimization and the sampled data split, so their variance is not purely optimization variance.

XOR3 has six inputs, of which only the first three affect its label. Mux4, Mux8, and Mux16 have 6, 11, and 20 inputs, respectively. `conjunction6` and `conjunction` are aliases, as are `sparse_dnfN` and `planted_dnfN`; aliases are counted once. Planted DNF16 has six manually specified clauses of length two or three. The 32-, 64-, and 100-dimensional planted tasks each have six three-literal clauses, with targets generated separately by dimension.

Table 5: Complete-catalogue results for the final method. Exact counts SAT-certified runs over all runs in the row. Fixed-task spreads are across five run seeds; random-group spreads are across teacher means. Each teacher has five runs. The three blocks are classical tasks, fixed constructions, and random teachers. Conjunction has no positive test examples, so balanced accuracy is undefined. Detailed per-task results appear in Table 6.

Benchmark / teacher seeds	d	Tasks	Acc. (%)	Bal. acc. (%)	Exact
XOR3	6	1	100.000 ± 0.000	100.000 ± 0.000	5/5
Parity6	6	1	100.000 ± 0.000	100.000 ± 0.000	5/5
Parity12	12	1	100.000 ± 0.000	100.000 ± 0.000	5/5
Parity16	16	1	100.000 ± 0.000	100.000 ± 0.000	5/5
Majority7	7	1	100.000 ± 0.000	100.000 ± 0.000	5/5
Conjunction	6	1	95.000 ± 11.180	–	3/5
Mux4	6	1	90.000 ± 13.693	91.667 ± 11.785	3/5
Mux8	11	1	100.000 ± 0.000	100.000 ± 0.000	5/5
Mux16	20	1	97.065 ± 2.870	97.068 ± 2.867	2/5
Nested logic	8	1	100.000 ± 0.000	100.000 ± 0.000	5/5
Planted DNF16	16	1	100.000 ± 0.000	100.000 ± 0.000	5/5
Planted DNF32	32	1	100.000 ± 0.000	100.000 ± 0.000	5/5
Planted DNF64	64	1	100.000 ± 0.000	100.000 ± 0.000	5/5
Planted DNF100	100	1	100.000 ± 0.000	100.000 ± 0.000	5/5
DNF16-C6-L3 (s0–2)	16	3	100.000 ± 0.000	100.000 ± 0.000	15/15
DNF32-C6-L3 (s0–2)	32	3	100.000 ± 0.000	100.000 ± 0.000	15/15
DNF32-C12-L3 (s0–2)	32	3	100.000 ± 0.000	100.000 ± 0.000	15/15
DNF64-C6-L3 (s0–2)	64	3	100.000 ± 0.000	100.000 ± 0.000	15/15
Mixed32-C3-L3-H3 (s0–2)	32	3	100.000 ± 0.000	100.000 ± 0.000	15/15
DNF32-C12-L3 (s100–109)	32	10	99.757 ± 0.338	99.846 ± 0.214	45/50
Mixed32-C3-L3-H3 (s100–109)	32	10	100.000 ± 0.000	100.000 ± 0.000	50/50
DNF32-C12-L3 (s130–139)	32	10	100.000 ± 0.000	100.000 ± 0.000	50/50
Mixed32-C3-L3-H3 (s130–139)	32	10	100.000 ± 0.000	100.000 ± 0.000	50/50

Target construction. Labels are deterministic and contain no injected noise. Parity tasks use the XOR of their stated inputs; Conjunction is the AND of all six inputs; Majority7 is positive when at least four inputs are true. A multiplexer uses its first $\log_2 m$ bits as a binary address selecting one of the following m data bits. With one-based indices, Nested logic is

$$[(x_1 \wedge \neg x_2) \oplus (x_3 \wedge x_4)] \vee [(x_5 \vee x_6) \wedge (x_7 \oplus x_8)]. \tag{22}$$

Planted DNF16 is the fixed formula

$$\begin{aligned}(x_1 \wedge \neg x_4 \wedge x_8) \vee (\neg x_2 \wedge x_5) \vee (x_3 \wedge x_6 \wedge \neg x_9) \\ \vee (x_7 \wedge \neg x_{10}) \vee (x_{11} \wedge x_{12} \wedge \neg x_{13}) \vee (\neg x_{14} \wedge x_{15} \wedge x_{16}).\end{aligned} \tag{23}$$

For the larger planted targets, NumPy seed $1729 + d$ draws six distinct signed three-literal clauses. Random DNF teacher sk instead uses seed $1009 + k$, drawing the stated number of clauses. Each clause samples three distinct features and independent Boolean signs; duplicate signed clauses are rejected, without guaranteeing a minimal DNF. Mixed teacher sk draws seven such clauses with seed $1009 + 2{,}000{,}003 + k$ and combines them in the overlap pattern of Appendix A.5. The run seed controls uniform sampling without replacement, with the sampled array partitioned into the three disjoint splits in Table 6.

Table 6: All 69 tasks. Each row uses five seeds; all spreads are sample standard deviations across those five runs. L is the sum of literal counts over distinct clauses. An asterisk marks task rows whose five entries retain complete end-to-end training artifacts.

Task	d	Train/val/test	Acc. (%)	Bal. acc. (%)	Exact	L
xor3	6	48/8/8	100.000 ± 0.000	100.000 ± 0.000	5/5	3.00 ± 0.00
parity6	6	48/8/8	100.000 ± 0.000	100.000 ± 0.000	5/5	6.00 ± 0.00
mux4	6	48/8/8	90.000 ± 13.693	91.667 ± 11.785	3/5	12.40 ± 1.52
mux8	11	1536/256/256	100.000 ± 0.000	100.000 ± 0.000	5/5	32.00 ± 0.00
conjunction*	6	48/8/8	95.000 ± 11.180	–	3/5	4.40 ± 2.61
nested_logic	8	192/32/32	100.000 ± 0.000	100.000 ± 0.000	5/5	17.20 ± 3.90
planted_dnf16	16	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	16.00 ± 0.00
planted_dnf32	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	18.40 ± 0.89
planted_dnf64	64	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	18.00 ± 0.00
planted_dnf100	100	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	18.00 ± 0.00
parity12	12	3072/512/512	100.000 ± 0.000	100.000 ± 0.000	5/5	15.00 ± 6.71
parity16*	16	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	16.00 ± 0.00
majority7*	7	96/16/16	100.000 ± 0.000	100.000 ± 0.000	5/5	140.00 ± 0.00
mux16	20	4096/1024/4096	97.065 ± 2.870	97.068 ± 2.867	2/5	76.40 ± 5.68
random_dnf16c6l3s0	16	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	18.00 ± 0.00
random_dnf16c6l3s1	16	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	18.00 ± 0.00
random_dnf16c6l3s2	16	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	18.00 ± 0.00
random_dnf32c6l3s0	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	18.00 ± 0.00
random_dnf32c6l3s1	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	18.00 ± 0.00
random_dnf32c6l3s2	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	18.00 ± 0.00
random_dnf32c12l3s0	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s1	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s2	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf64c6l3s0	64	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	18.00 ± 0.00
random_dnf64c6l3s1	64	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	18.00 ± 0.00
random_dnf64c6l3s2	64	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	18.00 ± 0.00
random_mixed32c3l3h3s0	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s1	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s2	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_dnf32c12l3s100	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s101	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s102	32	4096/1024/4096	99.360 ± 1.430	99.606 ± 0.881	4/5	35.40 ± 1.34
random_dnf32c12l3s103	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s104	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s105	32	4096/1024/4096	99.795 ± 0.459	99.854 ± 0.327	4/5	37.80 ± 4.02
random_dnf32c12l3s106	32	4096/1024/4096	99.736 ± 0.590	99.841 ± 0.355	4/5	36.00 ± 0.00
random_dnf32c12l3s107	32	4096/1024/4096	99.004 ± 2.227	99.365 ± 1.421	4/5	34.80 ± 2.68
random_dnf32c12l3s108	32	4096/1024/4096	99.678 ± 0.721	99.794 ± 0.460	4/5	35.40 ± 1.34
random_dnf32c12l3s109	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_mixed32c3l3h3s100	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s101	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s102	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s103	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s104	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s105	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.40 ± 0.89
random_mixed32c3l3h3s106	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s107	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s108	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00

Table 7: Recovered program structures, with identical counts across the five seeds of each task. All programs are target-equivalent. C counts distinct clauses, Length gives literals per clause, L counts total literal occurrences, and U counts distinct variables used.

Task	Head	C	Length	L	U
Parity16	XOR	16	1	16	16
Majority7	OR	35	4	140	7
Planted DNF100	OR	6	3	18	16

Task	d	Train/val/test	Acc. (%)	Bal. acc. (%)	Exact	L
random_mixed32c3l3h3s109	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_dnf32c12l3s130	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s131	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s132	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s133	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s134	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s135	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s136	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s137	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s138	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_dnf32c12l3s139	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	36.00 ± 0.00
random_mixed32c3l3h3s130	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s131	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s132	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s133	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s134	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s135	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s136	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s137	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s138	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00
random_mixed32c3l3h3s139	32	4096/1024/4096	100.000 ± 0.000	100.000 ± 0.000	5/5	21.00 ± 0.00

C.2 Structural audit of exported programs

Table 7 summarizes the saved programs for all five seeds of each of three selected tasks. We count distinct clauses, literal occurrences across those clauses, and distinct input variables separately; repeated use of a variable does not increase the last count. For Parity16, Majority7, and Planted DNF100, we inspect the final exports for all five run seeds. Every selected run has a saved target-equivalence certificate. The exporter verifies all 16 positive singleton clauses for Parity16, all 35 positive four-variable conjunctions for Majority7, and six three-literal clauses using 16 distinct variables and both literal signs for Planted DNF100. These are descriptive structural checks on selected tasks, not a comparison of program complexity across methods or an independent component ablation.

The catalogue-wide Random-DNF audit compares each exported signed clause with the corresponding teacher clause after sorting literals within clauses, deduplicating clauses, and ignoring head placement when it does not alter the outer-OR function. All 155 SAT-exact Random-DNF exports recover exactly the teacher's signed-clause set. Of these, 153 use one OR head; two place one teacher clause in a separate single-clause XOR head, which computes that clause unchanged. The five non-equivalent Random-DNF runs are excluded from the structural-match claim and are listed in Appendix C.7.

C.3 Training settings and implementation scope

Table 8 records the actual final configurations. The complete-run Logic-KAN environment records an NVIDIA A100-SXM4-40GB, Python 3.12.13, PyTorch 2.11.0+cu128, CUDA 12.8, and NumPy 2.4.6, with float32 computation and no automatic mixed precision. The catalogue runner partitions tasks over available GPUs, using one worker per nonempty task shard and one numerical CPU thread per worker. The recorded environment is not a measurement of the compute cost of a new 345-run end-to-end training campaign, so we make no catalogue-wide efficiency claim.

Neural DNF here uses the original semi-symbolic parameterization; the experiment does not include a separate node-disentanglement procedure. RRL and DiffLogic select hard-model checkpoints on validation data at the configured evaluation interval of ten epochs, also checking the final epoch.

Table 8: Configurations used in the retained results. External neural methods use 400 epochs and batch size 512. Search settings are method-specific and are not compute matched.

Method	Configuration
Logic-KAN	Adam, learning rate 0.03, batch size 256, 400 epochs; 96 clause slots, 8 heads, 6 literals per clause; temperature annealed from 1.0 to 0.3; local-edit budget 256 and at most 4 edit steps per event; editing stops at training fraction 0.90; at most 64 empirical minimization steps and 64 retained joint candidates.
Neural DNF	64 conjunctive units; Adam learning rate 0.01; L_1 coefficient 10^{-5}; 12 candidate discretization thresholds from weight-magnitude quantiles 0.35–0.90, selected on validation data.
RRL	64 conjunctive and 64 disjunctive units with a two-class linear readout; Adam learning rate 0.002; squared-weight penalty coefficient 0.001; gradient grafting and weights clipped to $[0, 1]$.
DiffLogic	Four logic layers, hidden width 256, 64 output gates, fixed sampled connectivity, and 16 candidate operators per gate; Adam learning rate 0.01; classification-logit temperature 8.
BRCG	AIX360 beam search with a SciPy HiGHS master LP; $\lambda_0 = \lambda_1 \in \{10^{-4}, 10^{-3}, 10^{-2}\}$, selected on validation; degree 4, beam width 10, up to 10 new columns per pricing call; iteration setting 50 and time-budget setting 60 seconds per regularization value (checked between iterations, not a strict wall-time cutoff).
BoolXAI	Official RuleClassifier; depth 4, complexity 48, 1000 iterations, 4 starts, regularization 10^{-4}; AND, OR, Choose, AtMost, and AtLeast operators.

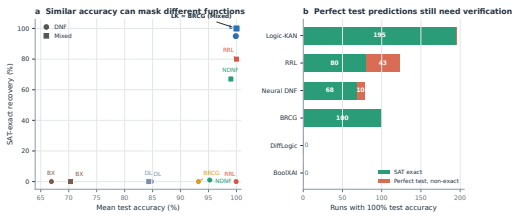


Figure 3: **Accuracy versus functional recovery.** Panel (a) places each method and teacher family by mean test accuracy and SAT-exact recovery. Panel (b) decomposes runs with 100% test accuracy into SAT-equivalent and perfect-test but non-equivalent exports. Counts use the two external teacher batches.

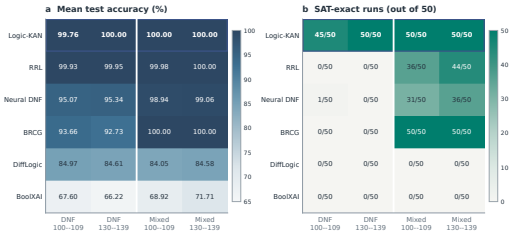


Figure 4: **External comparison by teacher batch.** The left panel reports mean test accuracy and the right panel reports SAT-exact recovery for the same four conditions as Table 3.

Neural DNF trains for the full epoch budget and selects its extraction threshold on validation data. BRCG selects its regularization setting by validation performance. BoolXAI returns the rule selected by its official training procedure. The neural baselines share a project training harness rather than the original authors' complete task-specific training pipelines.

C.4 Complete external metrics and verification semantics

Table 3 reports accuracy and exact recovery for both teacher batches. Table 9 adds balanced accuracy, F1, MCC, and soft–hard disagreement for every retained external condition. F1 and MCC are computed from hard predictions. For methods with a continuous output, Gap compares that output thresholded at 0.5 with the saved hard predictions on the test set. The RRL export rounds readout coefficients to integers at scale 10^6; its reported hard metrics and SAT outcomes refer to that exported artifact. They do not constitute a full-domain equivalence guarantee for an unrounded floating-point readout.

Table 9: Complete external comparison. Classification scores and disagreement are computed from saved predictions. Spreads are across ten teacher means, each averaging five runs. BRCG and BoolXAI have no continuous-model disagreement measurement.

Method	Acc. (%)	Bal. acc. (%)	F1 (%)	MCC	Exact	Gap (%)
DNF32-C12-L3; teacher seeds 100–109						
Logic-KAN	99.757 ± 0.338	99.846 ± 0.214	99.845 ± 0.220	0.993 ± 0.009	45/50	0.062 ± 0.142
RRL	99.934 ± 0.031	99.881 ± 0.064	99.959 ± 0.020	0.998 ± 0.001	0/50	0.193 ± 0.112
Neural DNF	95.069 ± 2.078	96.100 ± 1.893	96.811 ± 1.446	0.866 ± 0.046	1/50	4.940 ± 2.090
BRCG	93.662 ± 1.443	83.554 ± 4.061	96.218 ± 0.848	0.788 ± 0.053	0/50	–
DiffLogic	84.974 ± 1.036	64.662 ± 4.931	91.268 ± 0.688	0.423 ± 0.082	0/50	1.767 ± 0.627
BoolXAI	67.603 ± 2.144	68.242 ± 2.957	76.781 ± 1.784	0.298 ± 0.054	0/50	–
Mixed32-C3-L3-H3; teacher seeds 100–109						
Logic-KAN	100.000 ± 0.000	100.000 ± 0.000	100.000 ± 0.000	1.000 ± 0.000	50/50	0.000 ± 0.000
RRL	99.984 ± 0.028	99.980 ± 0.036	99.987 ± 0.023	1.000 ± 0.001	36/50	0.057 ± 0.109
Neural DNF	98.940 ± 1.121	99.149 ± 0.912	99.117 ± 0.952	0.979 ± 0.022	31/50	1.060 ± 1.121
BRCG	100.000 ± 0.000	100.000 ± 0.000	100.000 ± 0.000	1.000 ± 0.000	50/50	–
DiffLogic	84.046 ± 0.659	81.642 ± 1.449	87.633 ± 0.521	0.657 ± 0.019	0/50	3.292 ± 0.493
BoolXAI	68.921 ± 1.898	69.026 ± 1.531	72.919 ± 2.599	0.375 ± 0.031	0/50	–
DNF32-C12-L3; teacher seeds 130–139						
Logic-KAN	100.000 ± 0.000	100.000 ± 0.000	100.000 ± 0.000	1.000 ± 0.000	50/50	0.000 ± 0.000
RRL	99.947 ± 0.027	99.907 ± 0.036	99.967 ± 0.017	0.998 ± 0.001	0/50	0.106 ± 0.056
Neural DNF	95.336 ± 1.809	95.882 ± 2.380	97.022 ± 1.177	0.870 ± 0.047	0/50	4.679 ± 1.796
BRCG	92.729 ± 1.423	80.854 ± 4.214	95.709 ± 0.782	0.751 ± 0.058	0/50	–
DiffLogic	84.615 ± 0.508	62.763 ± 2.961	91.146 ± 0.433	0.396 ± 0.047	0/50	1.703 ± 0.396
BoolXAI	66.219 ± 2.748	67.482 ± 1.975	75.655 ± 2.383	0.282 ± 0.038	0/50	–
Mixed32-C3-L3-H3; teacher seeds 130–139						
Logic-KAN	100.000 ± 0.000	100.000 ± 0.000	100.000 ± 0.000	1.000 ± 0.000	50/50	0.143 ± 0.452
RRL	99.998 ± 0.006	99.998 ± 0.006	99.998 ± 0.005	1.000 ± 0.000	44/50	0.048 ± 0.146
Neural DNF	99.057 ± 1.348	99.195 ± 1.188	99.090 ± 1.464	0.983 ± 0.022	36/50	0.943 ± 1.348
BRCG	100.000 ± 0.000	100.000 ± 0.000	100.000 ± 0.000	1.000 ± 0.000	50/50	–
DiffLogic	84.577 ± 0.855	83.001 ± 1.503	87.561 ± 0.509	0.675 ± 0.023	0/50	2.715 ± 0.527
BoolXAI	71.706 ± 1.972	71.760 ± 1.706	75.073 ± 1.964	0.431 ± 0.036	0/50	–

The SAT query concerns the exported Boolean function and the known teacher, independently of the soft–hard disagreement statistic. The final report consolidation reused the saved target certificates rather than invoking the solver again. The manuscript audit recomputed saved prediction metrics for all 345 development-catalogue runs and 1000 external runs, replayed every final program on deterministically reconstructed test inputs, and checked the program–certificate hashes. The post-development integrity audit separately checked all 580 formal runs, including the twelve saved hidden-mismatch counterexamples. These audits did not retrain models or rerun the SAT solver. Pairing covers 200 task/run combinations, with the same frozen arrays used by all five external methods for each combination.

C.5 Joint-selection guards and the Mixed teacher

Joint program selection proposes affine-XOR rules, positive-example conjunction closures, and signed-count rules using training examples only. An affine system must be consistent and have a unique solution. Count candidates require a uniquely determined observed threshold and must fit within the existing clause and literal limits when expanded into DNF. A replacement must fit both training and validation labels exactly. If the checkpoint-selected program already fits both splits, it is retained except for a positive-closure candidate that strictly refines a single conjunction. With either training class absent, joint program selection is skipped. These rules specify an inductive preference, not a distribution-free correctness guarantee.

The Mixed teacher's seven-clause DNF identity is proved in Appendix A.5. Exact recovery of this family does not require reproducing its original OR/XOR head layout.

C.6 Post-development audit details

The post-development matrix contains 116 jobs and 580 complete runs. B1 is the 150-run blind complete-model block; B2 contains 400 mechanism-control runs on ten anchor teachers; and B3 contains 30 candidate-attribution runs. All 32-dimensional jobs use the same frozen 4096/1024/4096

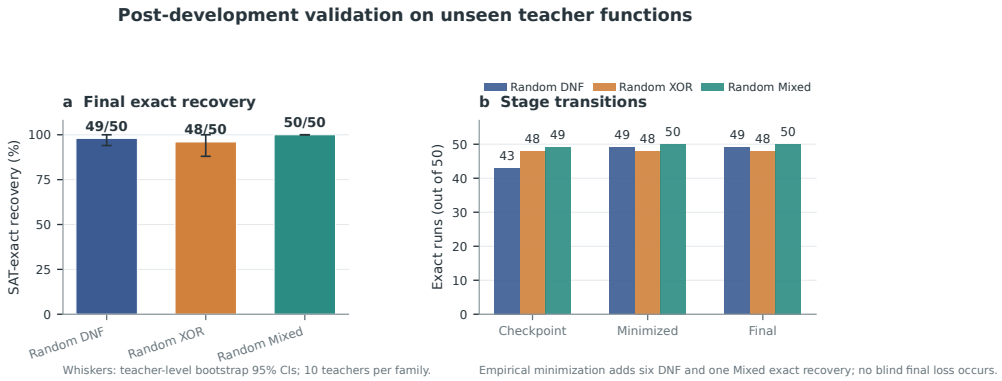


Figure 5: **Post-development validation on unseen teacher functions.** (a) Final SAT-exact recovery with teacher-level bootstrap intervals. (b) Exact recoveries at checkpoint, after empirical minimization, and after final selection. The 150 runs are all complete-model blind evaluations; no ablation controls are included.

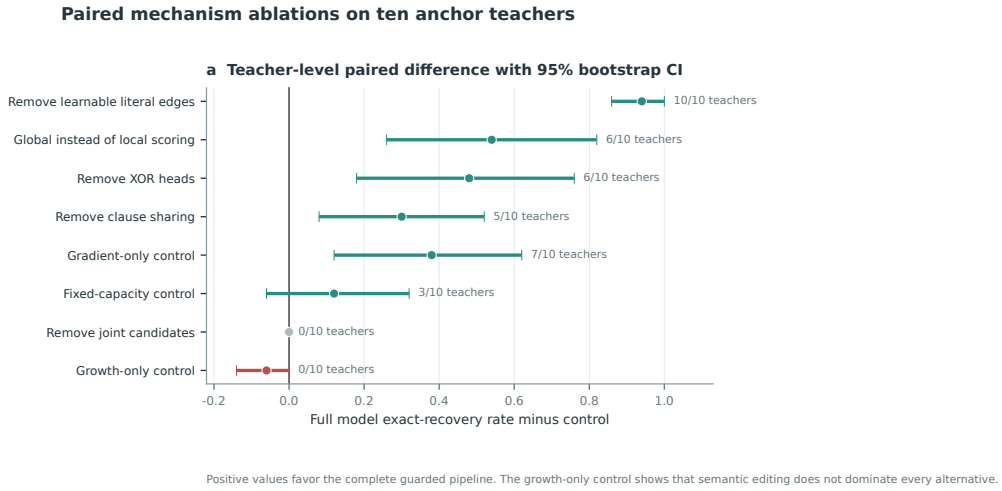


Figure 6: **Paired mechanism ablations.** Points and intervals show the complete pipeline's teacher-level exact-recovery rate minus that of each control, with bootstrap 95% intervals over ten teachers. Positive values favor the complete pipeline. The growth-only control is retained as a negative result; it prevents a claim that semantic editing dominates every alternative.

train, validation, and test arrays within each paired comparison. The integrity audit reports 580/580 complete runs with no missing predictions, unresolved SAT query, or hash mismatch. Nine early jobs with mixed or stale source snapshots were excluded from the reported matrix and rerun under the frozen source manifest; their archived outputs are not included in any count above.

The twelve perfect-test but SAT-non-equivalent runs are listed by block and variant in the retained formal CSV. Ten are gradient-only controls, one is a complete Conjunction run, and one is its no-closure control. They are not combined with the twelve failures in the development-catalogue table below: the latter includes runs with test errors and uses a different denominator.

C.7 Remaining failures and a current SAT witness

Table 12 lists every failed target-equivalence certificate in the development catalogue. The five DNF failures are the only failures among 275 random-teacher runs; teachers s103 and s109 recover in all five runs. All rows refer to the final Logic-KAN specification used for the development catalogue.

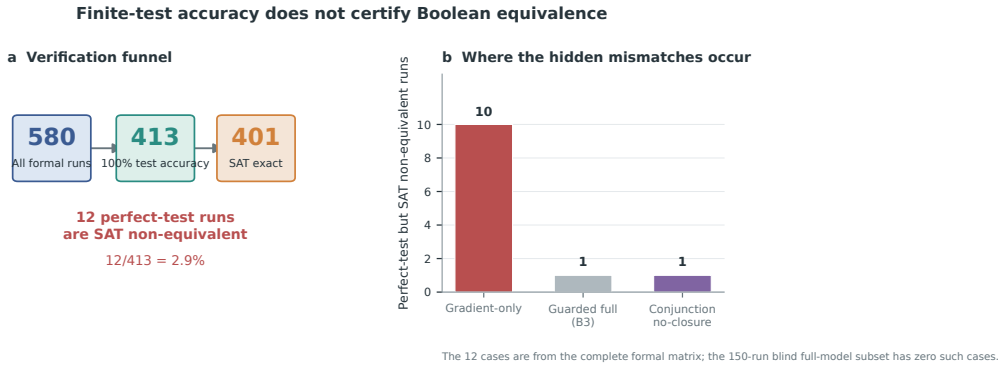


Figure 7: **Finite-test accuracy does not certify Boolean equivalence.** The funnel summarizes the complete 580-run formal matrix, which includes blind runs and controls. The right panel locates the twelve perfect-test but SAT-non-equivalent runs; the 150-run blind complete-model subset has zero such cases.

Table 10: Complete paired mechanism controls on ten anchor teachers. The full pipeline recovers 47/50 in every comparison. The difference is the teacher-level exact-recovery rate of the full pipeline minus the control; intervals are bootstrap 95% confidence intervals over ten teachers.

Control	Control exact	Difference	95% CI	Full better / tied / worse	Interpretation
Without joint candidates	47/50	0.00	[0.00, 0.00]	0/10/0	no gain on random anchors
Global scoring	20/50	+0.54	[0.26, 0.82]	6/4/0	local scoring matters
Frozen literal edges	0/50	+0.94	[0.86, 1.00]	10/0/0	ternary edges matter
OR-only heads	23/50	+0.48	[0.18, 0.76]	6/3/1	XOR supports parity
Unshared clauses	32/50	+0.30	[0.08, 0.52]	5/4/1	sharing supports composition
Gradient-only control	28/50	+0.38	[0.12, 0.62]	7/2/1	multi-mechanism control
Fixed-capacity control	41/50	+0.12	[−0.06, 0.32]	3/6/1	inconclusive capacity effect
Growth-only control	50/50	−0.06	[−0.14, 0.00]	0/8/2	no universal editing advantage

For Conjunction seed 4, the exported program has no retained heads and is therefore the constant-false function. All eight test labels are negative, giving 100% test accuracy. The saved SAT witness is $x = (1, 1, 1, 1, 1, 1)$: the program returns zero while the conjunction target returns one. This is the sole final main-run example with perfect test accuracy but failed target equivalence in the development catalogue. Its training split contains no positive example, so the joint-selection guard preserves the checkpoint-selected program.

C.8 Retained evidence and reproduction limits

The final main table uses one Logic-KAN result per task and run seed. Complete end-to-end training artifacts are retained for 15 entries; the remaining 330 entries retain the final predictions, programs, selection records, and certificates needed for result auditing. Source hashes for the paper tables are provided with the table-export script. The retained evidence includes final programs, predictions, structural-edit records, target-equivalence certificates, and complete artifacts for the 15 end-to-end runs. The external results retain 200 unique frozen splits and their SHA-256 hashes. The underlying gradient-training states for the full catalogue were removed during result consolidation, so the retained compact artifact does not support replaying every optimization trajectory end to end. The post-development artifact retains the 116-job manifest, final programs, predictions, stage certificates, source snapshots, and integrity hashes for all 580 formal runs. The new blind teachers are synthetic Boolean functions; the project still contains no public real-world dataset experiments. The reported blind evaluation therefore supports out-of-catalogue recovery within the evaluated teacher grammar, not generalization to real-world rule-learning tasks.

Table 11: Targeted candidate attribution. Each condition has five run seeds. Removing a candidate tests a structure-specific inductive bias, not a universal requirement of the grammar.

Task and candidate	Complete	Removed
Parity16, affine-XOR	5/5	0/5
Majority7, signed-count	5/5	0/5
Conjunction, positive closure	3/5	0/5

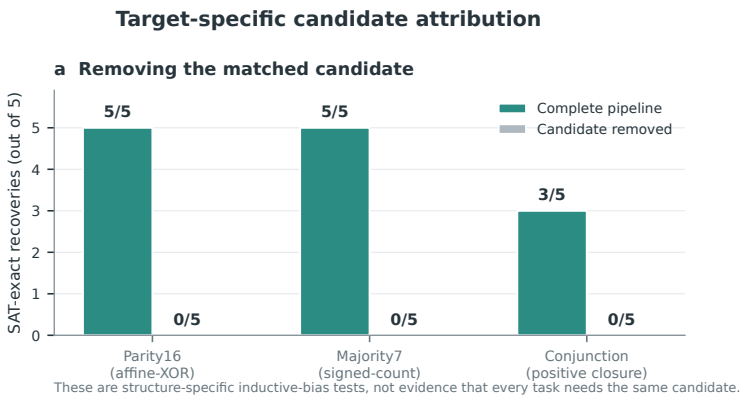


Figure 8: Target-specific candidate attribution. The candidate library provides useful inductive biases for parity, count, and conjunction structures; it is not a single mechanism required by every teacher.

Table 12: All 12 development-catalogue failures. Every row has a saved SAT counterexample. Errors count hard-program test disagreements.

Task	Run seed	Test errors	Test examples
Conjunction	0	2	8
Conjunction	4	0	8
Mux4	2	2	8
Mux4	4	2	8
Mux16	0	262	4096
Mux16	3	196	4096
Mux16	4	143	4096
DNF32-C12-L3, s102	3	131	4096
DNF32-C12-L3, s105	4	42	4096
DNF32-C12-L3, s106	1	54	4096
DNF32-C12-L3, s107	1	204	4096
DNF32-C12-L3, s108	2	66	4096

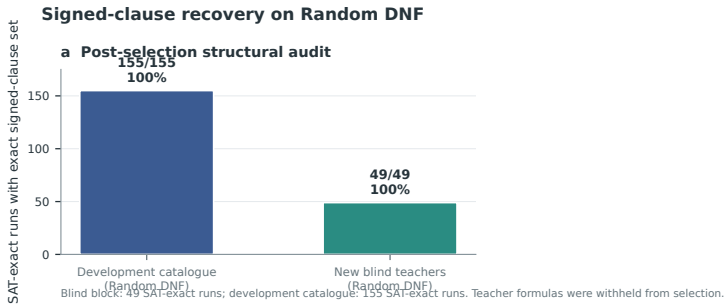


Figure 9: Random-DNF structural audit. Every SAT-exact export in the 155-run development catalogue and the 49-run blind block matches its teacher's signed-clause set. Teacher formulas are used only after model selection for this audit.