LEARNING TO CONTROL BDD SIZE

Anonymous authors
Paper under double-blind review

ABSTRACT

Binary Decision Diagrams (BDDs) are canonical representations of Boolean functions widely used to compactly and exactly represent and query large state spaces across computer science and artificial intelligence. Their principal limitation is a severe dependence on variable ordering, under which the same function can admit exponentially different representation sizes. Dynamic variable reordering (DVR) addresses this problem using hand-designed heuristics that search over orderings through sequences of adjacent variable swaps, often requiring substantial reordering effort to obtain high-quality solutions. We instead formulate DVR as a sequential decision-making problem and investigate whether good reordering decisions can be learned and generalized. First, we show that tabular reinforcement learning can discover orderings that closely match the strongest established heuristics while requiring only a small number of swaps at execution. We then distill these learned action values into a neural controller built around a compositional bottom-up encoder of BDD subgraphs and hierarchical attention that aggregates and contextualizes variable-level structure. The resulting policy generalizes to unseen Boolean functions and to larger variable counts than observed during training, retaining near-teacher reordering quality while using only a small fraction of the swaps required by competitive search-based heuristics. These results suggest that expensive online combinatorial search over symbolic representations can be substantially replaced through learned control.

1 INTRODUCTION

Binary Decision Diagrams (BDDs) are symbolic representations of Boolean functions used to represent and manipulate large state spaces exactly (Bryant, 1986; 1992). They have played an important role across computer science and artificial intelligence, including formal verification, model checking, and hardware synthesis (Burch et al., 1992; Malik et al., 1988; Bryant, 1986; Minato, 1996; Wille & Drechsler, 2009); knowledge compilation, symbolic reasoning, and planning (Darwiche & Marquis, 2011; Hoey et al., 1999; Bahar et al., 1993; Shih et al., 2018); probabilistic and numerical computation (Kwiatkowska et al., 2002; Chavira & Darwiche, 2008; Fujita et al., 1997); and program and network analysis, reliability, configuration, and combinatorial manipulation (Whaley et al., 2005; Yuan et al., 2006; Reay & Andrews, 2002; Hadzic et al., 2004; Minato, 1993). Their effectiveness, however, depends critically on representation size, which in turn depends critically on the variable ordering used to define the BDD's decomposition order. This sensitivity has made variable ordering a central problem in BDD research and led to dynamic variable reordering (DVR) (Rudell, 1993), a well-established collection of heuristics that search the permutation space by applying sequences of adjacent variable swaps that manipulate a live BDD directly. The main drawback of these methods is the large amount of online search they require, often involving many thousands of variable swaps.

In this paper we argue that DVR is particularly well suited to learned control via deep reinforcement learning: the live BDD exposes rich structural information about variable interactions, each swap changes that structure while providing an exact size-based reward, and the value of a swap depends on the values of future swaps it enables. This gives both a highly compositional and informative state representation as well as a natural sequential objective, allowing a neural controller to learn from and exploit structure that hand-designed heuristics cannot. The main aim of this paper is therefore to determine whether a BDD-specific neural encoding and policy architecture can learn from offline reordering experience and generalize that experience to unseen BDDs. A central objective is to do

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.

so without guidance from existing heuristics and without online search at inference time, thereby avoiding large-scale combinatorial search over complex symbolic structures.

We address this in three components. First, we exploit the permutation topology of DVR through archive-assisted tabular Q-learning (Watkins & Dayan, 1992), providing a cheap and stable way to generate high-quality per-instance reordering experience that serves as the teacher for the neural controller. Second, we introduce a BDD-specific neural architecture that follows the compositional structure of the diagram, first constructing bottom-up embeddings of Boolean subfunctions and then using hierarchical attention to summarize variable-level structure and model interactions across the ordering. Third, we use policy distillation (Rusu et al., 2016) to transfer the tabular teacher's reordering experience across many unrelated problem instances into a single neural controller capable of generalizing to unseen BDDs. The resulting neural controller performs remarkably well. On unseen BDDs, including functions with larger variable counts than those seen during training, it remains competitive with the strongest and most search-intensive heuristics, frequently matching and occasionally surpassing their ordering quality while executing only a tiny fraction of their swaps. Despite the additional neural inference overhead of our current prototype, there are already many test instances where achieving the controller's ordering quality through heuristic search requires substantially more wall-clock time. Given the dramatic reduction in reordering operations, there remains substantial scope for further runtime gains through highly optimized inference, tighter integration with the underlying BDD package, and further architectural and methodological improvements.

2 Related Work

Learning-based approaches have previously addressed the variable ordering problem, but have done so by predicting a single static ordering from a formula-level representation of the Boolean function before compiling it into a BDD (Grumberg et al., 2003; Miao et al., 2026). Reinforcement learning has also been applied in an adjacent setting, selecting variables one by one during the construction of approximate decision diagrams so as to tighten the resulting optimization bounds (Cappart et al., 2019; 2022). Also relevant is work on neural-guided SAT and QBF solving (Lederman et al., 2020; Kurin et al., 2020; Selsam & Bjørner, 2019; Tönshoff & Grohe, 2026), where learned models reduce combinatorial search by guiding branching decisions, a recurring challenge is that neural inference can offset these gains in wall-clock time, although increasingly sparse and tightly integrated guidance has progressively reduced this overhead and begun to yield practical speedups.

3 Background

3.1 Binary Decision Diagrams with Complemented Edges

A BDD $G_f^{\prec}$ is a rooted directed acyclic graph representing a Boolean function $f : \{0,1\}^n \to \{0,1\}$ under a fixed variable ordering $\prec$. Each non-terminal node v tests a Boolean variable x_v with a corresponding level $level(v)$ in the ordering and has two children: a high child u_h followed when $x_v = 1$ and a low child u_l followed when $x_v = 0$. Variables occur in increasing level order along every root-to-terminal path. We use the standard complemented-edge representation used by almost all BDD packages, with a single terminal node representing 1. Each branching node v is associated with a bit $c_v \in \{0,1\}$ indicating whether its low edge is complemented, high edges are always regular. Each subdiagram rooted at node v defines a Boolean subfunction defined recursively as $f_v = x_v f_{u_h} + \overline{x_v}\big(f_{u_l} \oplus c_v\big)$, with the terminal defining the constant function 1. A complement bit on the root similarly allows the complete diagram to represent either the root subfunction or its negation. Reduction rules merge equivalent subdiagrams and remove nodes with both edges pointing to the same child. This makes the representation recursively canonical, no (sub)function is represented more than once. Once constructed, many Boolean queries and transformations are tractable in the BDD size rather than the underlying 2^n valuation space. For instance satisfiability and tautology are decidable in $\mathcal{O}(1)$, and model counting and restriction require $\mathcal{O}(|G_f^{\prec}|)$. New BDDs are commonly constructed using a graph composition algorithm, which recursively combines two diagrams $G_f^{\prec}$ and $G_g^{\prec}$ sharing the same variable ordering $\prec$ under some Boolean operator with worst-case time and output size $\mathcal{O}(|G_f^{\prec}| \cdot |G_g^{\prec}|)$. Under the same ordering, equivalent functions will

always compile to the same BDD. For a concrete illustration of this representation, Appendix A provides an example BDD with complemented edges.

3.2 Dynamic Variable Reordering

BDD size can depend dramatically on the variable ordering, with the same Boolean function admitting exponentially different diagram sizes under different orderings. Finding an ordering $\prec$ minimizing $|G_f^{\prec}|$ is computationally hard, with the corresponding decision problem being NP-complete. A naive search over candidate orderings would require repeatedly reconstructing the BDD under each ordering, making the search cost depend on the potentially very large intermediate diagrams produced by unfavorable candidates. Dynamic variable reordering instead modifies an already constructed BDD in place, allowing alternative orderings to be explored without recompiling the Boolean function from scratch. The fundamental DVR primitive is the swap of two variables at adjacent levels. Swapping levels i and $i+1$ locally restructures the affected portion of the DAG while preserving the represented Boolean function and maintaining reduction. The operation is tractable in the current diagram size, requiring $\mathcal{O}(|G_f^{\prec}|)$ time in the worst case. Since any variable ordering can be reached through a sequence of adjacent swaps, established DVR methods build larger reordering procedures from this same local primitive. The most widely used DVR paradigm is sifting (Rudell, 1993), which considers variables in turn and repeatedly moves each variable through the ordering by adjacent swaps, recording the BDD size at each position and returning the variable to the best position encountered. Symmetric sifting extends this procedure by detecting symmetric neighboring variables and subsequently moving them together as a group (Panda et al., 1994), while group sifting generalizes the same idea by allowing groups to form based on broader structural affinity rather than exact symmetry (Panda & Somenzi, 1995). Converging variants repeat the corresponding reordering procedure until no further reduction is obtained. More global search strategies include simulated annealing (Bollig et al., 1996), which can accept worsening moves to escape local minima, and genetic algorithms (Drechsler et al., 1996), which evolve a population of candidate variable orderings.

4 DVR as a Markov Decision Process

For a fixed Boolean function f, canonicity to variable ordering gives DVR a finite search space over permutations and a natural deterministic Markov decision process (MDP) formulation. If $\prec_t$ denotes the variable ordering at step t and $G_t := G_f^{\prec_t}$ the corresponding BDD, we define the state as $S_t = (G_t, b_t)$, where b_t is the remaining reordering budget. Rather than acting through individual adjacent swaps, we define each non-terminal action as a source–destination pair $A_t = (i_t, j_t)$ that moves the variable currently at level i to level j, implemented by sliding it through the ordering via adjacent swaps. This permits long-range moves to be selected directly, avoiding the need to traverse potentially unfavorable intermediate orderings as separate decisions. A flat representation gives $\mathcal{O}(n^2)$ candidate moves, so we instead factor action selection into two $\mathcal{O}(n)$ decisions: first selecting a source i, then a destination j conditioned on that source. This reflects the natural distinction between identifying which variable should move and determining where it should be placed, while allowing experience across different destinations to inform a shared source-value estimate. We represent the two stages using action-value functions $Q_{\mathrm{src}}(S_t, i)$ and $Q_{\mathrm{dst}}(S_t, i, j)$. Source selection forms an intermediate decision context (S_t, i) without modifying the BDD, consuming budget, or introducing discount. With $\gamma = 1$, their Bellman targets are

$$Q_{\mathrm{src}}(S_t, i) \leftarrow \max_{j \neq i} Q_{\mathrm{dst}}(S_t, i, j), \qquad Q_{\mathrm{src}}(S_t, \textsc{Stop}) = 0,$$

$$Q_{\mathrm{dst}}(S_t, i, j) \leftarrow R_t + \max_k Q_{\mathrm{src}}(S_{t+1}, k).$$

After selecting a source i_t, choosing destination j_t executes the resulting composite move $A_t = (i_t, j_t)$, yielding $S_{t+1} = (G_{t+1}, b_t - 1)$. Selecting Stop terminates the episode immediately, and states with no remaining budget are terminal. We decompose the reward for each variable move into two components:

$$R_t = \underbrace{(|G_t| - |G_{t+1}|)}_{\text{size reward}} - \underbrace{(\lambda_s |i_t - j_t| + \lambda_m)}_{\text{move penalty}}$$

where $|G_t|$ denotes BDD size and $|i_t - j_t|$ is the number of adjacent swaps required to execute the move. The coefficients λ_s and λ_m respectively weight the cost of adjacent swaps and complete

variable moves. With $\gamma = 1$, the size rewards telescope over an episode to the overall initial-to-final size reduction and the move penalty favors shorter and fewer moves.

5 Discovering Reordering Policies with Tabular RL

5.1 Archive-Assisted Q-Learning

Canonicity makes DVR particularly amenable to tabular reinforcement learning, since states can be indexed by hashing the current ordering and remaining move budget rather than storing the BDD itself. Moreover, for a fixed number of variables, all instances share the same transition graph over orderings, differing only in their rewards and values. We exploit this common structure using archive-assisted tabular Q-learning to retain and reuse information gathered during exploration. We maintain a bounded archive $\mathcal{E}$ of previously reached endpoint orderings $\prec_e$ and their BDD sizes $|G_f^{\prec_e}|$. For a current ordering $\prec_t$ and archived endpoint $\prec_e$, the minimum movement cost is $C(\prec_t, \prec_e) = \lambda_m(n - LCS(\prec_t, \prec_e)) + \lambda_s D(\prec_t, \prec_e)$, where LCS denotes longest common subsequence length and $D(\prec_t, \prec_e)$ is the minimum adjacent-swap distance. For a candidate action (i, j) producing successor ordering $\prec_{t+1}$, its archive-assisted value is

$$Q_{\text{arch}}(S_t, i, j) = \max_{\prec_e \in \mathcal{E}} \left[|G_t| - |G_f^{\prec_e}| - \lambda_s |i - j| - \lambda_m - C(\prec_{t+1}, \prec_e) \right],$$

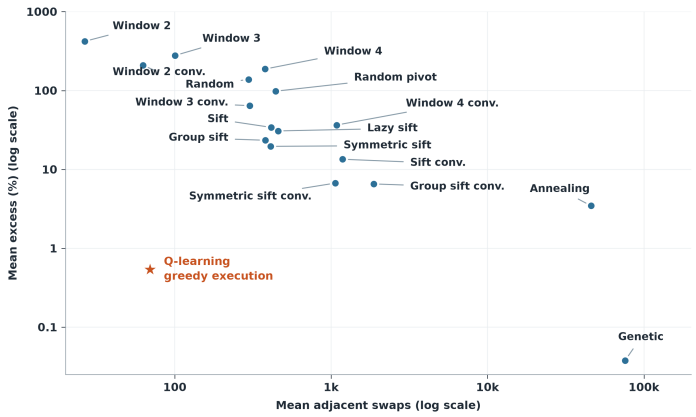
where the maximum is taken only over archived endpoints reachable from $\prec_{t+1}$ within the remaining move budget. These values can be computed from permutation distances and stored endpoint sizes alone, without constructing or evaluating the intermediate BDDs along each route. The bounded archive is updated throughout training as better orderings are discovered while maintaining diversity among retained endpoints. Training uses independent ϵ-greedy decisions at the source and destination stages for exploration, while greedy choices use the larger of the learned and archive-derived action values:

$$i_t = \arg\max_i \left[\max_{j \neq i} \lceil \max\{Q_{\text{dst}}(S_t, i, j), Q_{\text{arch}}(S_t, i, j)\} \rceil \right],$$
$$j_t = \arg\max_{j \neq i_t} \lceil \max\{Q_{\text{dst}}(S_t, i_t, j), Q_{\text{arch}}(S_t, i_t, j)\} \rceil .$$

The ceiling operation is used only for training-time action selection, grouping values that differ only by small movement penalties so that actions with similar estimated size reduction continue to be explored, stored Q-values and Bellman backups remain unrounded. Bellman backups are applied online in reverse through each completed episode. STOP is excluded from action selection during training and its value is automatically backed up for each visited state. When a source–destination action is executed through adjacent swaps, intermediate destinations encountered along the move also provide observed rewards that are backed up. Training begins with regular ϵ-greedy exploration and later biases exploration toward promising regions by first following the current greedy route to the best ordering discovered so far before resuming exploration. To reduce training cost, we cache the measured size of each visited ordering and avoid keeping the underlying BDD synchronized with the agent state, lazily reordering it only when an unseen ordering must be evaluated. Full tabular-training hyperparameters, archive settings, exploration schedules, penalty coefficients, and move-budget details are provided in Appendix D.

5.2 Tabular Results

Throughout this paper, we use a dataset of Boolean functions farmed from public benchmarks, including ISCAS (Brglez & Fujiwara, 1985; Brglez et al., 1989), LGSynth (Yang, 1991), IWLS (McElvain, 1993; Albrecht, 2005), VTR (Rose et al., 2012), QUIP (Malhotra et al., 2004), ICCAD (Hsu et al., 2015; Huang et al., 2017), VCEGAR (Jain et al., 2007), and Verilog2SMV (Irfan et al., 2016). We derive 56,805 Boolean functions spanning 10 to 30 variables by extracting subfunctions from the larger benchmark circuits, with at least 2,000 functions at each variable count. Full extraction, filtering, and deduplication details are provided in Appendix D. For the initial tabular-discovery pass, each function is initialized from a random variable ordering. The resulting start BDD sizes average 28.8 nodes at 10 variables, with a maximum of 155, up to 596.4 nodes at 30 variables, with a maximum of 16,209. Figure 1 reports results across the full 56,805-function dataset. Additional



Method	Mean swaps	Mean excess (%)
Window 2	26.6	420.378
Window 3 conv.	300.9	64.348
Sift	413.7	34.088
Lazy sift	457.3	30.739
Sym. sift conv.	1061.6	6.700
Window 4 conv.	1082.8	36.481
Sift conv.	1183.9	13.437
Group sift conv.	1872.4	6.538
Annealing	45894.4	3.452
Genetic	75786.8	0.038
Q-learn. train	3.94×10^6	0.539
Q-learn. greedy	69.3	0.539

Figure 1: Mean adjacent swaps and mean excess over the best final BDD size obtained by the 17 heuristic baselines. Lower-left is better.

distributional statistics and ablation results for the tabular experiments are provided in Appendix C and B. Tabular Q-learning consistently discovers high-quality reordering policies whose greedy execution is competitive with the strongest heuristics while requiring only a small number of swaps. Although discovering these policies requires extensive search during training, it provides a stable mechanism for generating strong per-instance reordering experience, but does not yet generalize to other problem instances. This motivates the neural controller introduced next, which aims to consolidate and generalize this experience across unseen BDDs.

6 Neural Controller

6.1 Compositional Bottom-Up BDD Encoder

We first encode the current BDD G_t with a compositional bottom-up pass, computing a d-dimensional representation $\mathbf{z}_v \in \mathbb{R}^d$ for each node v that summarizes the subdiagram rooted at that node. For each level i, we compute a learned level embedding $\mathbf{l}_i = \psi(\frac{i}{n-1}, \log n)$, where ψ is a shared multi-layer perceptron (MLP). The embedding $\mathbf{l}_i$ provides a continuous representation of level i within the current variable ordering and is reused throughout the architecture as its positional identifier. The unique terminal node $\top$ initializes the bottom-up recursion with a learned embedding $\mathbf{e}_\top \in \mathbb{R}^d$, where $\mathbf{z}_\top = \mathbf{e}_\top$. For each branching node v with high child u_h, low child u_l, and low-edge complement bit $c_v \in \{0, 1\}$, we compute $\mathbf{z}_v = \phi\big(\mathbf{z}_{u_h} \| \mathbf{z}_{u_l} \| c_v \| \mathbf{l}_{level(v)}\big)$, where $\|$ denotes concatenation and ϕ is a shared MLP applied to every branching node. The ordered child embeddings preserve the distinction between the high and low branches, while c_v captures low-edge complementation and $\mathbf{l}_{level(v)}$ supplies the node's position within the current ordering. Since all children occur at deeper BDD levels, embeddings are computed once from the terminal toward the root, requiring $\mathcal{O}(|G_t|)$ node updates. Sharing ϕ across nodes and diagrams allows the encoder to operate on BDDs of arbitrary size and structure, while the level-ordered structure permits nodes whose children have already been encoded to be processed in parallel. From a representation-learning perspective, each BDD provides supervision not only for the represented function, but for the full collection of subfunctions in its recursive decomposition. Since every node roots a reduced BDD for one of these subfunctions, the shared encoder is forced to learn useful representations throughout the decomposition. When the same subfunctions recur in other BDDs, their canonical reduced structure allows those representations to be reused directly, providing a strong inductive bias toward transferable functional representations.

6.2 Hierarchical Level Transformer

The bottom-up encoder provides a representation for every subdiagram, but DVR acts on variables, with each variable occupying a single BDD level. The controller must therefore reason about the collective structure associated with each level and about interactions between levels across the ordering.

Direct global attention over all node embeddings would be prohibitively expensive for large BDDs, so we perform this reasoning hierarchically. We first compress the nodes at each level into a single learned summary $\mathbf{s}_i$. Let $\mathbf{z}_{v_0}, \ldots, \mathbf{z}_{v_m}$ denote the embeddings of nodes at level i. A learned query $\mathbf{q}$ acts as the query token and attends to the node embeddings, which serve as keys and values, through a shared cross-attention Transformer $\mathcal{T}_{\text{intra}}$ based on standard multi-head attention (Vaswani et al., 2017). This produces a single level summary, $\mathbf{s}_i = \mathcal{T}_{\text{intra}}\left(\mathbf{q}, [\mathbf{z}_{v_0}; \ldots; \mathbf{z}_{v_m}]\right)$. The use of a learned query to summarize an unordered collection of embeddings is related to attention-based pooling in Set Transformer (Lee et al., 2019). Because no ordering is imposed over nodes within a level, the aggregation is permutation-invariant to their enumeration while allowing the learned query to selectively weight individual subdiagrams. With a single query attending over the nodes of a level, the attention cost is linear in the number of nodes rather than quadratic as in full self-attention, making aggregation practical even for very large BDD levels.

The learned summary is complemented by structural information computed directly from the BDD. For each level i, we concatenate $\mathbf{s}_i$ with a vector $\mathbf{x}_i \in \mathbb{R}^{55}$ of structural features, including terminal-edge fractions, edge-span statistics, node-sharing fractions, and complemented-edge fractions. We then project the result back to dimension d and add the corresponding level embedding, $\mathbf{u}_i = \rho(\mathbf{s}_i \| \mathbf{x}_i) + \mathbf{l}_i$, where ρ is a shared projection. This combines learned subdiagram information with inexpensive structural statistics while preserving each level's position in the current ordering. The full feature set is given in the appendix. Finally, the level representations are processed jointly by an inter-level Transformer $\mathcal{T}_{\text{inter}}$, $[\mathbf{h}_0; \ldots; \mathbf{h}_{n-1}] = \mathcal{T}_{\text{inter}}\left([\mathbf{u}_0; \ldots; \mathbf{u}_{n-1}]\right)$. This confines quadratic self-attention to the much smaller sequence of n variable levels rather than the BDD nodes. Each $\mathbf{h}_i$ therefore represents the variable at level i in the context of the full ordering, allowing the controller to model how its reordering value depends on its relationship to variables at other levels. This final self-attention stage is particularly well-suited to DVR, where the value of moving a variable depends not only on its own substructure but on its relationship to variables throughout the ordering.

6.3 Factored Policy Heads

The contextualized level representations are mapped to actions using the same source–destination factorization introduced in the MDP formulation. To make decisions sensitive to the remaining horizon, we normalize the move budget as $\bar{b}_t = \frac{b_t}{n-1}$. A shared source-scoring MLP g_{src} independently assigns a score to each variable level, $s_i = g_{\text{src}}(\mathbf{h}_i \| \mathbf{l}_i \| \bar{b}_t)$, while the STOP logit is fixed to $s_{\text{STOP}} = 0$. The level embedding gives each variable an explicit positional identity, while the fixed zero logit provides a natural reference corresponding to the zero return of terminating reordering. Conditioned on selecting source level i, a second shared MLP g_{dst} scores each candidate destination, $d_{i,j} = g_{\text{dst}}(\mathbf{h}_i \| \mathbf{l}_i \| \mathbf{h}_j \| \mathbf{l}_j \| \bar{b}_t)$. Invalid destinations such as $j = i$ are masked. At inference, the controller greedily selects the highest-scoring source or STOP and, when a source is selected, the highest-scoring valid destination. These scores parameterize the factored policy and are trained by distillation from teacher action values as described in the following section.

6.4 Policy Distillation

We train the neural controller using policy distillation (Rusu et al., 2016) from the tabular teacher's action values. This provides a natural way to consolidate strong instance-specific policies into a single neural controller that can generalize across BDDs, while avoiding the much greater cost and instability of direct neural Q-learning, which would require repeatedly encoding large intermediate BDDs throughout exploratory trajectories. Rather than directly regressing Q-values, we use temperature-scaled cross-entropy to preserve the teacher's relative action preferences. For the source and destination decisions, respectively, the soft distillation losses are

$$\mathcal{L}_{\text{src,soft}} = -\sum_i \text{softmax}\left(\frac{Q_{\text{src}}(S,\cdot)}{\tau}\right)_i \log \text{softmax}([s_0; \ldots; s_{n-1}; 0])_i \,,$$

$$\mathcal{L}^{(i)}_{\text{dst,soft}} = -\sum_{j \neq i} \text{softmax}\left(\frac{Q_{\text{dst}}(S,i,\cdot)}{\tau}\right)_j \log \text{softmax}([d_{i,0}; \ldots; d_{i,n-1}])_j \,.$$

The final zero source logit corresponds to STOP, while $j = i$ is masked from the destination distribution. The temperature τ controls the sharpness of the teacher targets. We additionally apply

hard source and destination losses, $\mathcal{L}_{\mathrm{src,hard}}$ and $\mathcal{L}_{\mathrm{dst,hard}}$, using cross-entropy against the teacher-optimal actions and assigning uniform target mass when multiple actions are exactly tied. Because the policy is factored, we also supervise the complete action: for a teacher-optimal pair (i^*, j^*),

$$\mathcal{L}_{\mathrm{joint}} = -\log \mathrm{softmax}([s_0; \ldots; s_{n-1}; 0])_{i^*} - \log \mathrm{softmax}([d_{i^*,0}; \ldots; d_{i^*,n-1}])_{j^*} .$$

If STOP is uniquely optimal, only the source term is used. Exact global ties are handled by averaging the loss over the tied optimal complete actions. Combining these terms, the per-state training objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{src,soft}} + \mathcal{L}_{\mathrm{dst,soft}} + \lambda_h \left(\mathcal{L}_{\mathrm{src,hard}} + \mathcal{L}_{\mathrm{dst,hard}} + \mathcal{L}_{\mathrm{joint}} \right),$$

where the destination terms denote the mean of their corresponding conditional destination losses, and λ_h controls the additional emphasis placed on exact teacher-optimal actions.

7 EXPERIMENTS

7.1 EXPERIMENTAL SETUP

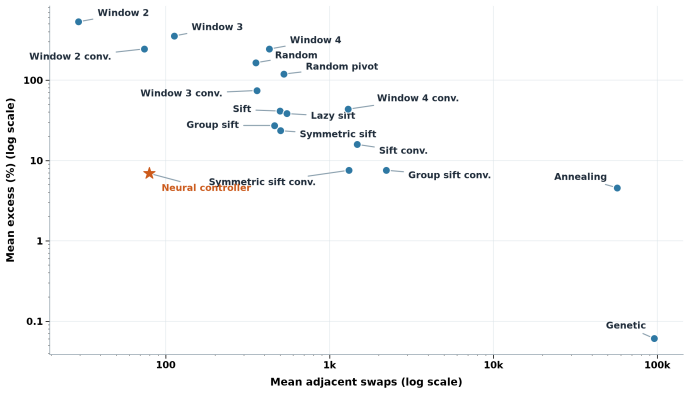
For neural target generation, we reuse the 56,805-function corpus spanning 10–30 variables introduced in the tabular validation. For each function, we retain the tabular value tables, endpoint archive, and ordering caches obtained from its original start ordering, then augment these with 49 additional random start orderings and run 500 tabular-training episodes from each. These shorter runs reuse the strong value estimates and endpoint orderings already discovered for the function, allowing rapid adaptation to new regions of the same ordering space. We extract supervision from the resulting greedy trajectories. At each visited state, dense destination targets are generated for four source variables, always including the teacher-greedy source, using archive-based continuation values and additional rollout episodes where required to refine the estimates. Restricting dense supervision to a small subset of sources keeps target generation tractable, since exhaustively producing high-quality targets for all source–destination pairs would require evaluating $\mathcal{O}(n^2)$ candidate moves at every state. This yields approximately 117.4M training and 12.5M validation state–budget examples. Full implementation details, including hyperparameter settings and function extraction, are provided in Appendix D. The BDD package we used was CUDD (Somenzi, 2015).

To evaluate generalization, we use the same benchmark collection and function-extraction procedure described above to construct an additional test set of 100 functions at each variable count from 10 to 40. To prevent leakage through related extractions, we audit the test set against the training corpus and exclude any function whose ancestor function in the original benchmark also contributed a subfunction to training. This yields 2,100 held-out functions within the 10–30 variable range seen during training and 1,000 functions with 31–40 variables for zero-shot extrapolation beyond the training range. To additionally test out-of-domain generalization, we evaluate on 500 functions extracted from the Aralia fault-tree analysis dataset (Earthperson et al., 2026), providing a distinct reliability-analysis domain from the hardware and electronic-design benchmarks used for training, which primarily span circuit synthesis, testing, verification, and model checking. The Aralia functions likewise span 10–40 variables, with 11–17 functions at each variable count, comprising 346 functions within the training size range and 154 functions with 31–40 variables. The benchmark-derived evaluation set spans a wide range of initial BDD sizes: at 10 variables, the starting diagrams contain 28.1 nodes on average with a maximum of 72, increasing to 546.1 nodes on average with a maximum of 6,810 at 25 variables, and 65,330.5 nodes on average with a maximum of 6,034,008 at 40 variables. The Aralia functions likewise exhibit substantial size variation, with an overall mean initial BDD size of 587.2 nodes and a maximum of 31,858 nodes. Additional details on distributions of initial BDD sizes are provided in Appendix C.

7.2 EXPERIMENTAL RESULTS

Figure 2 provides strong evidence for the central hypothesis of this work: high-quality dynamic variable reordering can be distilled into a greedy learned controller rather than recovered through extensive online search. Across held-out functions, zero-shot extrapolation to larger variable counts, and an unseen fault-tree domain, the controller remains competitive with the strongest established heuristics while requiring only a small fraction of their adjacent swaps. The effect is particularly striking given that inference performs no search over candidate orderings, but commits directly to each source–destination move from the learned policy.

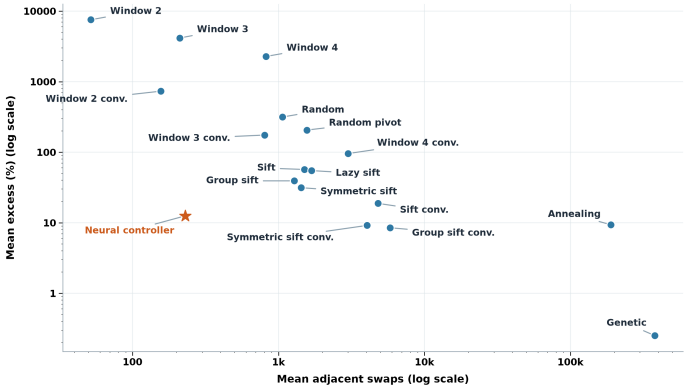
Held-out 10–30 variable evaluation



Method	Mean swaps	Mean excess (%)
Window 2	29.3	532.7
Window 3	112.9	355.3
Window 4	428.7	244.3
Sift	498.5	41.3
Lazy sift	548.1	38.4
Sift cnv.	1472.0	15.9
Sym. sift cnv.	1312.1	7.6
Grp sift cnv.	2218.8	7.6
Annealing	56984.3	4.6
Genetic	95870.2	0.1
Neural	**80.1**	**6.9**

	W2	W3	W4	Sift	Lazy	Sift cnv.	Sym. cnv.	Grp cnv.	Anneal.	Genetic	**Neural**
Runtime (ms), mean / median	0.5/0.3	0.6/0.4	0.8/0.7	0.8/0.8	0.9/0.8	1.3/1.2	1.3/1.2	1.8/1.6	32.7/26.9	55.1/43.9	**28.5/21.1**

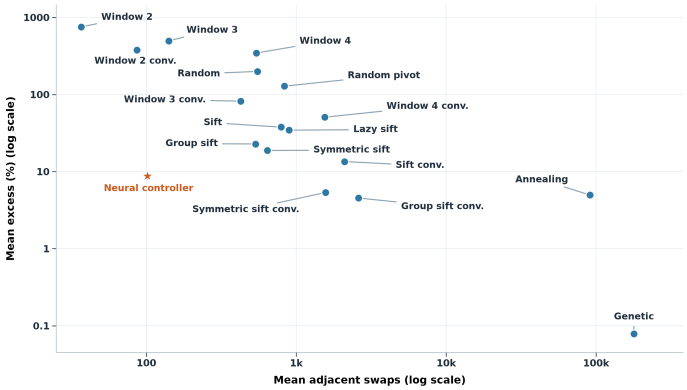
Extrapolation to 31–40 variables



Method	Mean swaps	Mean excess (%)
Window 2	51.9	7549.6
Window 3	210.8	4160.1
Window 4	820.9	2278.1
Sift	1505.1	57.0
Lazy sift	1688.7	54.9
Sift cnv.	4801.9	18.8
Sym. sift cnv.	4038.7	9.2
Grp sift cnv.	5809.9	8.5
Annealing	189549.3	9.4
Genetic	378968.4	0.3
Neural	**231.9**	**12.5**

	W2	W4	Sift	Lazy	Sift cnv.	Sym. cnv.	Grp cnv.	Anneal.	Genetic	**Neural**
Runtime (ms), mean / median	180.1/0.4	513.7/0.8	52.2/1.1	51.9/1.3	53.1/2.4	53.2/2.1	54.1/2.8	140.6/84.1	249.9/186.0	**423.0/71.5**

OOD transfer to Aralia fault-tree functions



Method	Mean swaps	Mean excess (%)
Window 2	36.8	751.6
Window 3	140.8	496.4
Window 4	540.8	344.5
Sift	792.6	37.7
Lazy sift	895.2	34.5
Sift cnv.	2092.6	13.5
Sym. sift cnv.	1564.6	5.3
Grp sift cnv.	2601.7	4.5
Annealing	91004.1	5.0
Genetic	178738.0	0.1
Neural	**102.1**	**8.8**

	W2	W3	W4	Sift	Lazy	Sift cnv.	Sym. cnv.	Grp cnv.	Anneal.	Genetic	**Neural**
Runtime (ms), mean / median	0.3/0.3	0.4/0.3	0.9/0.5	0.9/0.6	1.0/0.7	1.5/1.0	1.3/0.8	1.8/1.1	45.8/29.9	101.5/63.2	**39.2/15.8**

Figure 2: Mean adjacent swaps against mean excess over the per-instance best size obtained by the 17 existing heuristics. Lower-left is better.

Beyond the aggregate trends the per-instance results show that the controller frequently recovers the heuristic optimum exactly. Across the 3,100 benchmark-derived test functions, the neural controller matches the best final BDD size obtained by any of the 17 heuristics on 1,423 instances and improves on it on a further 11. On these 11 wins, the resulting BDDs are 4.16% smaller on average than the best heuristic result, with the strongest improvement reaching 7.41%. The best-seen and terminal BDD differ on only 181 functions (5.8%). The learned STOP action is selected before exhausting the move budget on 2,945 functions (95.0%), and in 2,849 of these cases (96.7%) the terminal BDD is already the best encountered. Performance also remains strong on the largest diagrams: among the ten largest starting BDDs in the 31–40 variable evaluation, including an instance with more than six million nodes, mean excess is only 7.5%, and five instances exactly match the best heuristic result. Poor outcomes are confined to a small tail: only 62 functions (2.0%) exceed 50% excess, seven exceed 100%, and the worst case reaches 201% excess. Similar instance-level behaviour is observed under domain shift. On the 500 Aralia functions, the controller exactly matches the best final BDD obtained by any of the 17 heuristics on 355 instances (71.0%), yielding a median excess of 0%; this includes 285 of 346 functions (82.4%) within the 10–30 variable training range and 70 of 154 (45.5%) when domain shift is combined with extrapolation to 31–40 variables. The best-seen and terminal BDD differ on only 27 functions (5.4%), while STOP is selected before exhausting the move budget on 476 functions (95.2%), with the terminal BDD already the best encountered in 467 of these cases (98.1%). Runtime exhibits a heavier-tailed distribution than swap count, with a small number of very large instances substantially skewing the mean; we therefore report median alongside mean runtime. The large reduction in reordering operations does not yet translate into an equally large reduction in wall-clock time, due to the neural encoding and inference cost. Nevertheless, the controller already achieves practical wins on many instances: it is faster than the fastest heuristic attaining at least the same solution quality on 303 functions, with a median speedup of $1.87\times$. These wins occur against annealing and genetic search. These results suggest substantial remaining headroom, since the present system is still a research prototype and runtime could likely be improved through tighter integration with the BDD package, more efficient inference, and further architectural and implementation optimization. Additional statistics on the distributions of swap count, excess BDD size, and runtime are provided in Appendix C.

8 CONCLUSION

This work provides strong evidence that one of the central bottlenecks in BDD-based computation—the expensive online search required to control representation size—can be substantially reduced through learned control. A greedy neural controller trained from offline reordering experience generalizes across unseen functions, larger variable counts, and a distinct application domain while recovering much of the quality of strong search-based heuristics using only a small fraction of their reordering operations. More broadly, the results show that useful reordering decisions can be predicted directly from BDD structure, opening the door to richer forms of representation control whose importance goes beyond wall-clock runtime. One important direction is to optimize not only the final ordering, but the path taken through permutation space. In practice, a promising ordering may be unreachable because intermediate BDDs grow so large that reordering stalls in time or memory. A controller trained to regulate transient growth could instead navigate toward strong orderings while avoiding these blow-ups, potentially reaching regions of the search space that conventional DVR cannot explore effectively. The same idea extends naturally to BDD construction itself. Compilation under a fixed ordering can fail before the final BDD is ever produced because intermediate diagrams become prohibitively large. Rather than treating reordering as a reactive repair step, a controller could reason over the current intermediate BDDs together with upcoming composition operations and reorder proactively to regulate growth throughout compilation. We therefore view the present results not only as a route to cheaper DVR, but as evidence for a broader learned-control framework in which variable ordering becomes an active part of the symbolic computation itself. A further issue for future work is scalability, particularly in the tabular stage. Tabular RL proved extremely useful here and need not be abandoned, but it will likely require stronger guidance as problem size grows. One natural extension is to bias tabular exploration with a neural policy or value model, reducing the number of states that must be visited and stored. Another possibility is to exploit BDDs themselves as compact symbolic representations of large value tables, which is already a well established approach (Hoey et al., 1999).

AI USE STATEMENT

GPT-5.6 Sol and GPT-6 Astra, accessed through ChatGPT and Codex, were used for writing assistance and research execution. In the main paper, writing assistance was predominantly local and sentence-level: the authors supplied the underlying text and ideas, and AI was used to improve wording, clarity, and concision, never generating more than approximately one paragraph at a time. The implementation-details section of the appendix was an exception and was largely drafted by GPT-5.6 Sol after inspecting the corresponding code. All technical statements, hyperparameters, and numerical values in this section were subsequently checked manually by the authors. Codex was used extensively to accelerate implementation and experimentation within an existing, author-developed and independently validated CUDD-based BDD control environment providing compilation, state construction, reordering actions, reward computation, and evaluation. It was particularly useful for rapidly running and analysing ablations, including experiments used to determine the final set of 55 structural features. All code used for the reported experiments was inspected, tested, and verified by the authors. AI systems were also used during research ideation. However, their suggestions were largely of limited value for the conceptually novel aspects of the work and led toward approaches that were subsequently rejected, particularly in neural architecture design. The central research direction, MDP formulation, and final architecture emerged from the authors' own ideation and investigation rather than from AI-generated proposals.

REPRODUCIBILITY STATEMENT

We provide the complete implementation of the proposed method in the supplementary material, including the CUDD-based BDD environment, archive-assisted tabular learner, teacher-target generation, neural architecture and training objective, structural-feature computation, and native C++/LibTorch evaluation pipeline. The supplement also includes the exact epoch-6 model weights used for the reported neural results, the exact 3,100 benchmark-derived and 500 Aralia evaluation functions with their starting variable orderings, provenance and checksums, and per-instance result tables underlying the reported aggregate statistics. The paper and Appendix D specify the data-construction procedure, training and validation split sizes, hyperparameters, target-generation procedure, architecture, optimization schedule, and inference protocol. The full 56,805-function training corpus and approximately 130 million generated training targets are not included in the submission due to their size. The provided model weights, evaluation data, and implementation are sufficient to rerun the reported held-out evaluations.

REFERENCES

C. Albrecht. Iwls 2005 benchmarks. Technical report, June 2005.

R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pp. 188–191, 1993. doi: 10.1109/ICCAD.1993.580054.

Beate Bollig, Martin Löbbing, and Ingo Wegener. On the effect of local changes in the variable ordering of ordered decision diagrams. *Information Processing Letters*, 59(5):233–239, 1996. ISSN 0020-0190. doi: https://doi.org/10.1016/0020-0190(96)00119-6. URL `https://www.sciencedirect.com/science/article/pii/0020019096001196`.

Franc Brglez and Hideo Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 663–698, 1985.

Franc Brglez, David Bryan, and Krzysztof Kozminski. Combinational profiles of sequential benchmark circuits. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 3, pp. 1929–1934, 1989.

Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986. doi: 10.1109/TC.1986.1676819.

Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, September 1992. ISSN 0360-0300. doi: 10.1145/136035.136043. URL https://doi.org/10.1145/136035.136043.

J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 1020 states and beyond. *Information and Computation*, 98(2):142–170, 1992. ISSN 0890-5401. doi: https://doi.org/10.1016/0890-5401(92)90017-A. URL https://www.sciencedirect.com/science/article/pii/089054019290017A.

Quentin Cappart, Emmanuel Goutierre, David Bergman, and Louis-Martin Rousseau. Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1443–1451, 2019. doi: 10.1609/aaai.v33i01.33011443.

Quentin Cappart, David Bergman, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Augustin Parjadis. Improving variable orderings of approximate decision diagrams using reinforcement learning. *INFORMS Journal on Computing*, 34(5):2552–2570, 2022. doi: 10.1287/ijoc.2022.1194.

Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6):772–799, 2008. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2007.11.002. URL https://www.sciencedirect.com/science/article/pii/S0004370207001889.

Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *CoRR*, abs/1106.1819, 2011. URL http://arxiv.org/abs/1106.1819.

Rolf Drechsler, Bernd Becker, and Nicole Göckel. Genetic algorithm for variable ordering of OBDDs. *IEE Proceedings - Computers and Digital Techniques*, 143(6):364–368, 12 1996. doi: 10.1049/ip-cdt:19960789.

Arjun Earthperson, Egemen Aras, Asmaa Salem Farag, Hasibul Hossain Rasheeq, Mihai Diaconeasa, and OpenPRA. openpra-org/aralia-fault-tree-dataset, May 2026. URL https://doi.org/10.5281/zenodo.20160659.

M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. *Formal Methods in System Design*, 10(2):149–169, Apr 1997. ISSN 1572-8102.

Orna Grumberg, Shlomi Livne, and Shaul Markovitch. Learning to order bdd variables in verification. *J. Artif. Int. Res.*, 18(1):83–116, January 2003. ISSN 1076-9757.

Tarik Hadzic, Sathiamoorthy Subbarayan, Rune Møller Jensen, Henrik Reif Andersen, Jesper Møller, and Henrik Hulgaard. Fast backtrack-free product configuration using a precompiled solution space representation. In *Proceedings from the International Conference on Economic, Technical and Organisational Aspects of Product Configuration Systems*, pp. 133–140. Technical University of Denmark, 2004. ISBN 8791035139.

Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. Spudd: stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, pp. 279–288, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1558606149.

Chih-Jen Hsu, Chi-An Wu, Wei-Hsun Lin, and Kei-Yong Khoo. ICCAD-2015 CAD contest in large-scale equivalence checking and function correction and benchmark suite. In Diana Marculescu and Frank Liu (eds.), *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2015, Austin, TX, USA, November 2-6, 2015*, pp. 916–920. IEEE, 2015. doi: 10.1109/ICCAD.2015.7372670. URL https://doi.org/10.1109/ICCAD.2015.7372670.

Ching-Yi Huang, Chih-Jen Hsu, Chi-An Wu, and Kei-Yong Khoo. ICCAD-2017 CAD contest in resource-aware patch generation. In Sri Parameswaran (ed.), *2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2017, Irvine, CA, USA, November 13-16, 2017*,

pp. 857–862. IEEE, 2017. doi: 10.1109/ICCAD.2017.8203868. URL https://doi.org/10.1109/ICCAD.2017.8203868.

Ahmed Irfan, Alessandro Cimatti, Alberto Griggio, Marco Roveri, and Roberto Sebastiani. Verilog2smv: A tool for word-level verification. In Luca Fanucci and Jürgen Teich (eds.), *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, pp. 1156–1159. IEEE, 2016. URL https://ieeexplore.ieee.org/document/7459485/.

Himanshu Jain, Daniel Kroening, Natasha Sharygina, and Edmund M. Clarke. VCEGAR: verilog counterexample guided abstraction refinement. In Orna Grumberg and Michael Huth (eds.), *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4424 of *Lecture Notes in Computer Science*, pp. 583–586. Springer, 2007. doi: 10.1007/978-3-540-71209-1_45. URL https://doi.org/10.1007/978-3-540-71209-1_45.

Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Can q-learning with graph networks learn a generalizable branching heuristic for a sat solver? In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 9608–9621. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6d70cb65d15211726dcce4c0e971e21c-Paper.pdf.

Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: Probabilistic symbolic model checker. In *Computer Performance Evaluation: Modelling Techniques and Tools*, volume 2324 of *Lecture Notes in Computer Science*, pp. 200–204. Springer, 2002. doi: 10.1007/3-540-46029-2_13.

Gil Lederman, Markus N. Rabe, Sanjit Seshia, and Edward A. Lee. Learning heuristics for quantified boolean formulas through reinforcement learning. In *8th International Conference on Learning Representations (ICLR)*, April 2020.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3744–3753. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/lee19d.html.

Shawn Malhotra, Terry P. Borer, Deshanand P. Singh, and Stephen Dean Brown. The quartus university interface program: enabling advanced FPGA research. In Oliver Diessel and John Williams (eds.), *Proceedings of the 2004 IEEE International Conference on Field-Programmable Technology, Brisbane, Australia, December 6-8, 2004*, pp. 225–230. IEEE, 2004. doi: 10.1109/FPT.2004.1393272. URL https://doi.org/10.1109/FPT.2004.1393272.

Sharad Malik, Albert R. Wang, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 6–9, 1988.

K. McElvain. Iwls'93 benchmark set: Version 4.0. Technical report, May 1993.

Mingkai Miao, Jianheng Tang, Guangyu Hu, and Hongce Zhang. Bdd2seq: Enabling scalable reversible-circuit synthesis via graph-to-sequence learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 863–872, 2026. doi: 10.1609/aaai.v40i2.37054.

Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pp. 272–277, 1993. doi: 10.1145/157485.164890.

Shin-ichi Minato. *Binary Decision Diagrams and Applications for VLSI CAD*. Kluwer Academic Publishers, 1996. ISBN 9780792396529.

S. Panda and F. Somenzi. Who are the variables in your neighbourhood. In *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, pp. 74–77, 1995. doi: 10.1109/ICCAD.1995.479994.

Shipra Panda, Fabio Somenzi, and Bernard F. Plessier. Symmetry detection and dynamic variable ordering of decision diagrams. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '94, pp. 628–631, Washington, DC, USA, 1994. IEEE Computer Society Press. ISBN 0897916905.

Karen A. Reay and John D. Andrews. A fault tree analysis strategy using binary decision diagrams. *Reliability Engineering & System Safety*, 78(1):45–56, 2002. doi: 10.1016/S0951-8320(02)00107-2.

Jonathan Rose, Jason Luu, Chi Wai Yu, Opal Densmore, Jeffrey Goeders, Andrew Somerville, Kenneth B. Kent, Peter Jamieson, and Jason Helge Anderson. The VTR project: architecture and CAD for fpgas from verilog to routing. In Katherine Compton and Brad L. Hutchings (eds.), *Proceedings of the ACM/SIGDA 20th International Symposium on Field Programmable Gate Arrays, FPGA 2012, Monterey, California, USA, February 22-24, 2012*, pp. 77–86. ACM, 2012. doi: 10.1145/2145694.2145708. URL https://doi.org/10.1145/2145694.2145708.

R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pp. 42–47, 1993.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy Distillation. In *International Conference on Learning Representations*, 2016. URL https://mlanthology.org/iclr/2016/rusu2016iclr-policy/.

Daniel Selsam and Nikolaj S. Bjørner. Guiding high-performance SAT solvers with unsat-core predictions. In Mikolás Janota and Inês Lynce (eds.), *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pp. 336–353. Springer, 2019. doi: 10.1007/978-3-030-24258-9_24. URL https://doi.org/10.1007/978-3-030-24258-9_24.

Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining bayesian network classifiers. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pp. 5103–5111. AAAI Press, 2018. ISBN 9780999241127.

Fabio Somenzi. Cudd: Cu decision diagram package, release 3.0.0, 2015.

Jan Tönshoff and Martin Grohe. Learning from algorithm feedback: One-shot sat solver guidance with gnns. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 98621–98642, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/a04c50c0bd6937760a3b89ee10010c96-Paper-Conference.pdf.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992. ISSN 1573-0565.

John Whaley, Dzintars Avots, Michael Carbin, and Monica S. Lam. Using datalog with binary decision diagrams for program analysis. In *Programming Languages and Systems*, volume 3780 of *Lecture Notes in Computer Science*, pp. 97–118. Springer, 2005. doi: 10.1007/11575467_8.

Robert Wille and Rolf Drechsler. Bdd-based synthesis of reversible logic for large functions. In *Proceedings of the 46th Annual Design Automation Conference (DAC)*, pp. 270–275, 2009. doi: 10.1145/1629911.1629984.

Saeyang Yang. Logic synthesis and optimization benchmarks user guide, version 3.0. Technical report, Microelectronics Center of North Carolina, 1991.

Lihua Yuan, Jianning Mai, Zhendong Su, Hao Chen, Chen-Nee Chuah, and Prasant Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, SP '06, pp. 199–213, USA, 2006. IEEE Computer Society. ISBN 0769525741. doi: 10.1109/SP.2006.16. URL https://doi.org/10.1109/SP.2006.16.

A BDD With Complemented Edges Example

Figure 3 gives a concrete example of the complemented-edge BDD representation used throughout this work. It represents

$$f(a,b,c,d,e) = \big(a \wedge (b \oplus c)\big) \vee \big(\neg a \wedge ((b \wedge d) \oplus (c \wedge e))\big)$$

under the variable ordering

$$a < b < c < d < e.$$

Solid edges denote high branches and dashed edges denote low branches. Hollow circles mark complemented references. Because negation is represented through complemented edges, only a single terminal node representing 1 is required. Shared subdiagrams are represented once, illustrating the reduced DAG structure of the BDD.

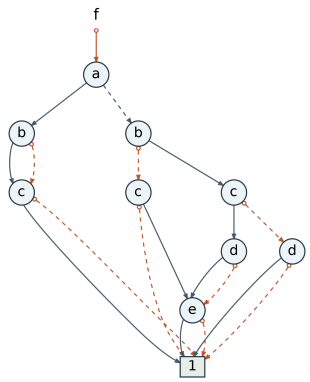


Figure 3: A reduced ordered BDD with complemented edges for the function above under ordering $a < b < c < d < e$. Solid and dashed edges denote high and low branches, respectively; hollow circles denote complementation. Shared descendants are represented once, and a single terminal represents 1.

B Ablations

B.1 Tabular RL

We ablate the archive-assisted training procedure by comparing the proposed method against standard tabular Q-learning using ordinary ϵ-greedy exploration, while keeping the remaining learning parameters and training budget fixed. Results are averaged over 100 functions and three matched random seeds per function. Regular Q-learning obtains a mean excess of 13.53%, whereas the archive-assisted method reduces this to 0.75%, demonstrating that the additional archive guidance substantially improves the quality of the policies discovered under the same episode budget. Figure 4 illustrates how this difference develops during training on one of the more difficult functions in the ablation set, plotting excess against training episode for regular Q-learning and the archive-assisted variant.

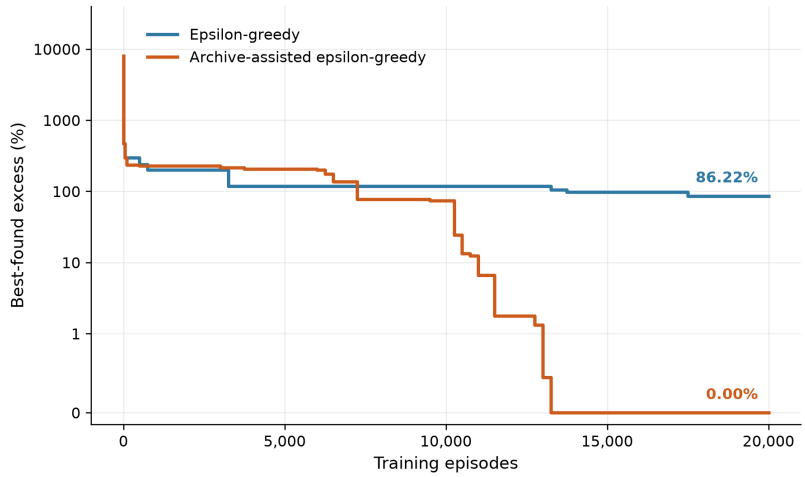


Figure 4: Example training progression on a difficult function from the tabular ablation set. Curves show excess BDD size as training proceeds for standard tabular Q-learning and the proposed archive-assisted variant.

B.2 Neural Controller

We ablate the main architectural components on a fixed 200-function structural stress panel spanning 21–30 variables, with 20 functions at each variable count. The panel was selected by initial BDD size and ancestry rather than by model performance, and all variants use the same training data, objective, and optimization setup except for the component being ablated. We report single-episode terminal mean excess over the best of the 17 heuristic baselines; lower is better.

The *mean pooling* variant replaces the learned within-level cross-attention with simple averaging, testing whether selective aggregation of node embeddings is useful. *No explicit features* removes the 55 handcrafted structural features, testing whether the learned BDD representations are sufficient on their own. *Features only* removes the learned within-level node representation pathway and retains the explicit structural features with inter-level attention, testing the contribution of the bottom-up learned representation. Finally, *no inter-level Transformer* retains the per-level summaries and explicit features but removes contextual reasoning across variable levels.

The full architecture performs best after the first epoch and remains the strongest variant through the remainder of training. Replacing cross-attention with mean pooling causes a moderate degradation, while removing the explicit features produces a larger loss, indicating that the learned and handcrafted representations provide complementary information. Using the explicit features without the learned node-representation pathway degrades performance further. The largest deterioration occurs when inter-level self-attention is removed, supporting the need to model interactions between variables across the complete ordering.

C Additional Statistics

In the distribution plots below, violins show the empirical density, embedded boxes show the median and interquartile range with $1.5\times$ IQR whiskers, and diamonds denote arithmetic means. To display heavy-tailed distributions while retaining resolution near zero, excess axes use an $\operatorname{asinh}(x/10)$ transformation, while BDD size, swap count, and runtime axes use $\log_{10}(1+x)$ scaling. Tick labels are reported in the original units.

C.1 Initial BDD Size Distributions

Figures 5–7 show the distributions of initial BDD sizes as a function of variable count for the training corpus, benchmark-derived evaluation set, and Aralia out-of-domain evaluation set, respectively.

Architecture	Epoch 1 excess (%)	Epoch 2 excess (%)	Epoch 3 excess (%)	Epoch 4 excess (%)	Epoch 5 excess (%)
Full model	75.9	**42.7**	**37.8**	**36.1**	**31.0**
Mean pooling	**59.9**	49.3	45.8	58.6	36.9
No explicit features	89.1	70.3	47.2	44.6	39.6
Features only	90.1	75.5	65.5	72.4	55.5
No inter-level Transformer	151.7	123.5	171.1	92.6	152.7

Table 1: Architecture ablations on the fixed 200-function structural stress panel. Entries report single-episode terminal mean excess over the best heuristic result; lower is better.

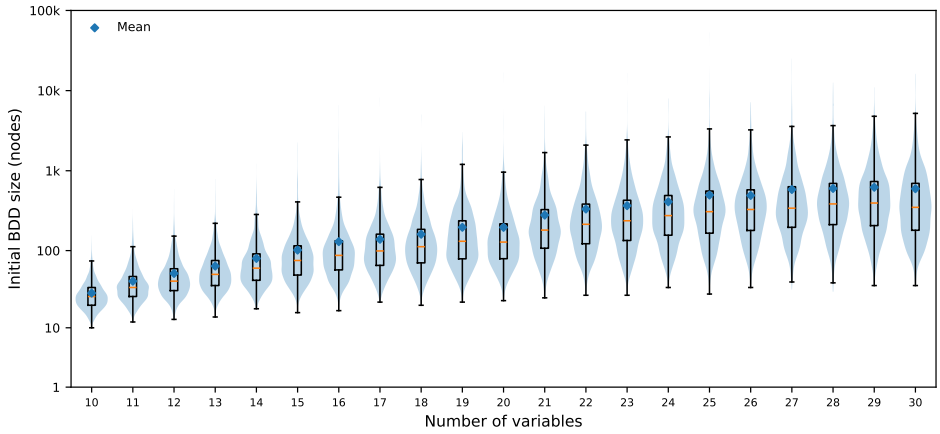


Figure 5: Distribution of initial BDD sizes in the 56,805-function training corpus as a function of variable count.

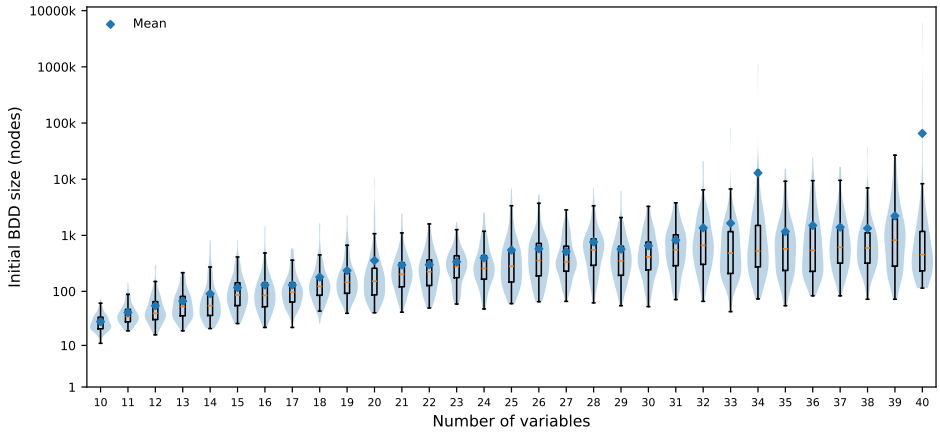


Figure 6: Distribution of initial BDD sizes in the 3,100-function benchmark-derived evaluation set. Each variable count contains 100 functions.

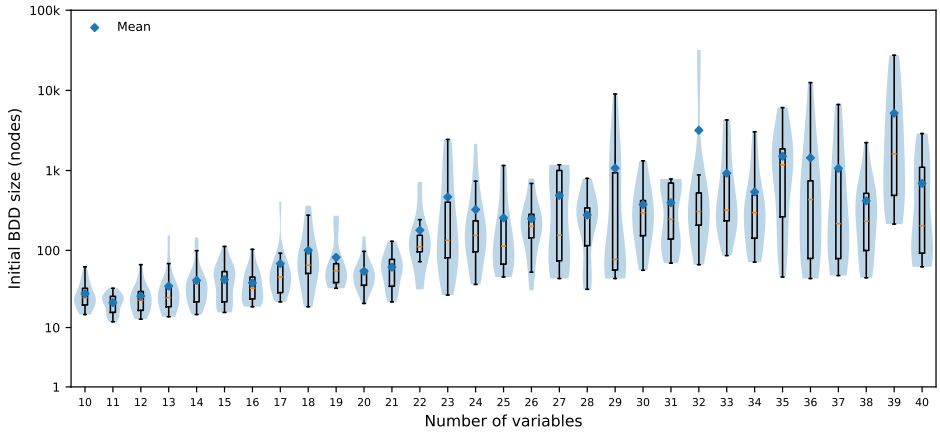


Figure 7: Distribution of initial BDD sizes in the 500-function Aralia out-of-domain evaluation set.

C.2 TABULAR RL

Figures 8 and 9 show the full per-instance distributions of excess BDD size and greedy execution swaps for the archive-assisted tabular policy and all 17 heuristic baselines across the 56,805-function dataset.

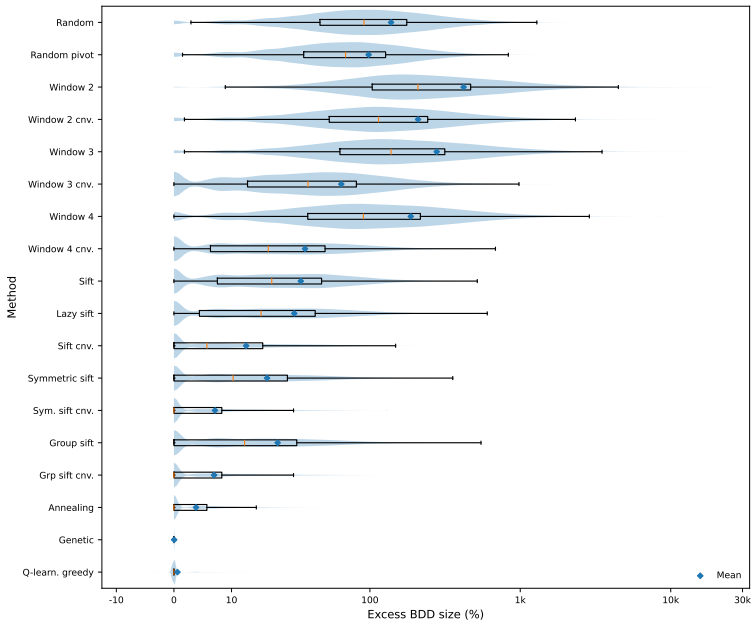


Figure 8: Distribution of excess BDD size over the best final BDD obtained by the 17 heuristic baselines across the full 56,805-function tabular dataset. The Q-learning result corresponds to greedy execution of the learned tabular policy.

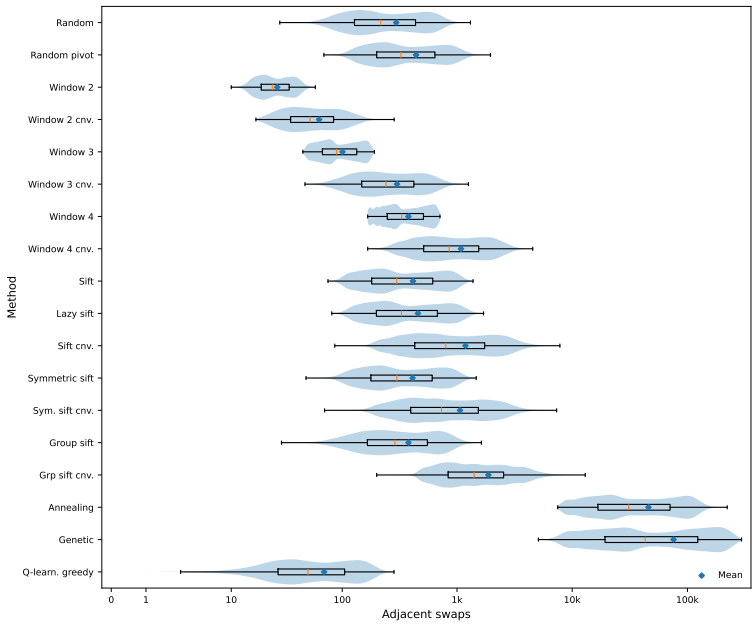


Figure 9: Distribution of adjacent-swap counts across the full 56,805-function tabular dataset. The Q-learning result reports swaps used by greedy policy execution rather than the much larger number explored during training.

C.3 Neural Controller

The following figures provide the corresponding per-instance distributions for the neural controller and all 17 heuristic baselines, separately for the benchmark-derived and out-of-domain evaluation sets.

C.3.1 Benchmark-Derived Test Set

Figures 10–12 show the distributions of excess BDD size, executed adjacent swaps, and reordering runtime across the 3,100 benchmark-derived test functions.

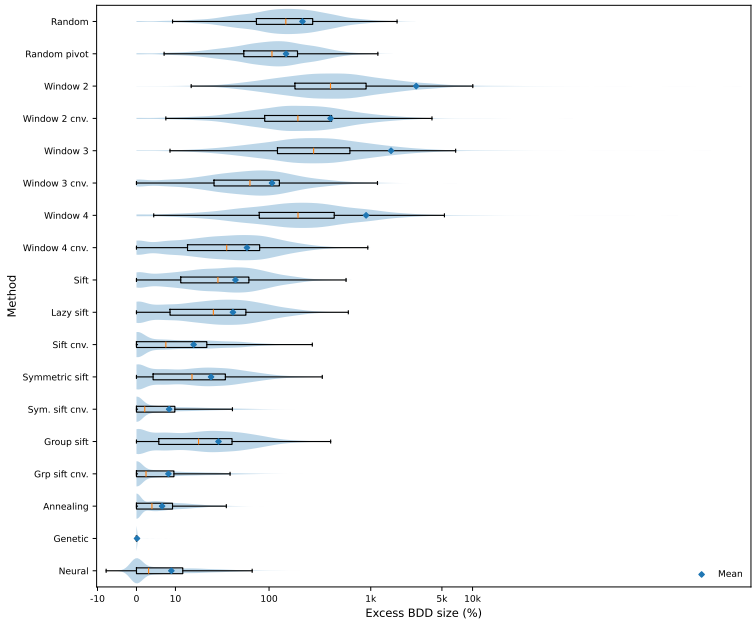


Figure 10: Distribution of excess BDD size across the 3,100 benchmark-derived test functions for the neural controller and all 17 heuristic baselines.

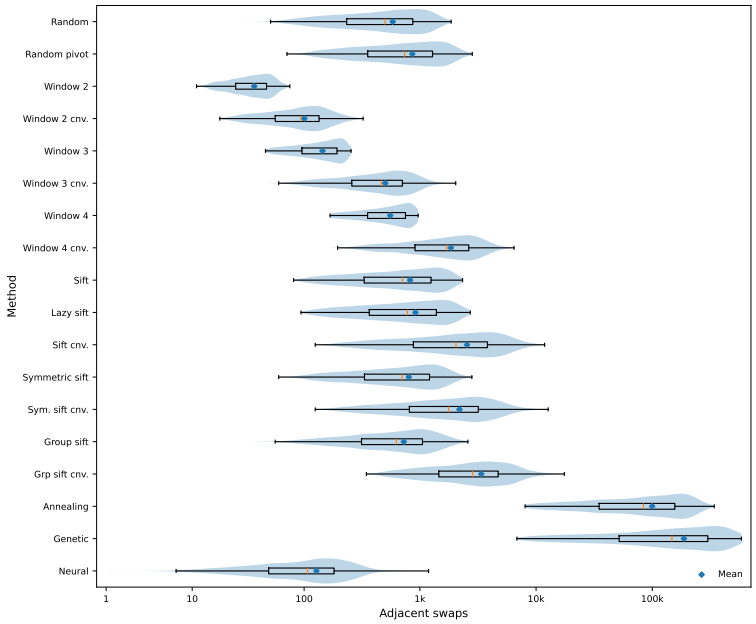


Figure 11: Distribution of adjacent-swap counts across the 3,100 benchmark-derived test functions for the neural controller and all 17 heuristic baselines.

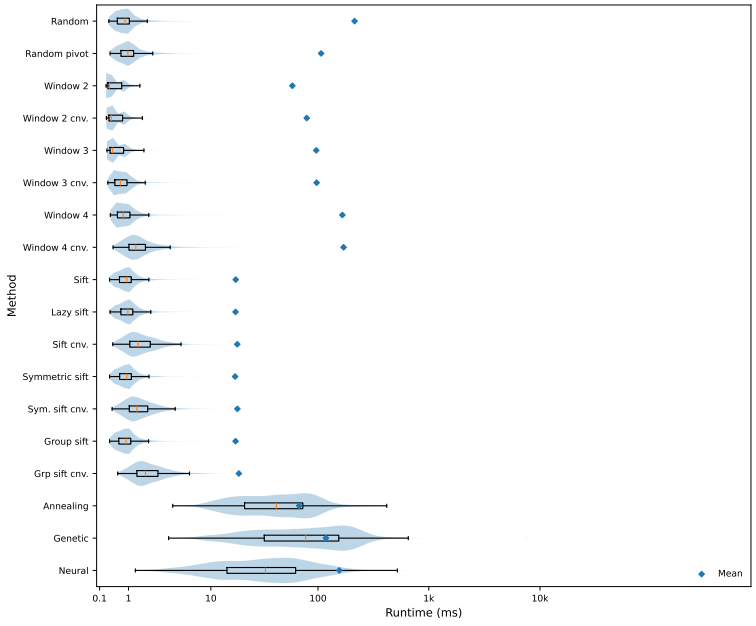


Figure 12: Distribution of reordering runtime across the 3,100 benchmark-derived test functions. The heavy upper tails explain the difference between the mean and median runtimes reported in the main text.

C.3.2 Aralia Out-of-Domain Test

Figures 13–15 show the same excess, swap-count, and runtime distributions for the 500 Aralia fault-tree functions, providing the corresponding statistics under out-of-domain transfer.

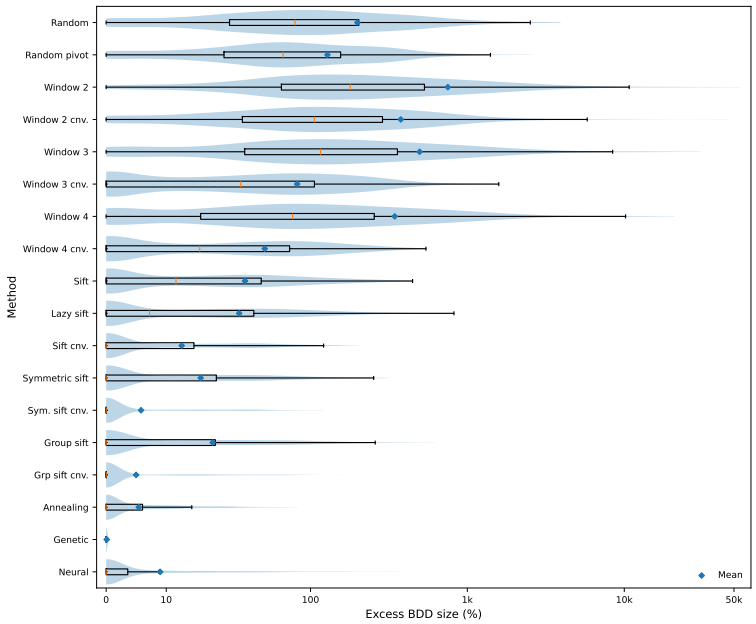


Figure 13: Distribution of excess BDD size across the 500 Aralia fault-tree functions for the neural controller and all 17 heuristic baselines.

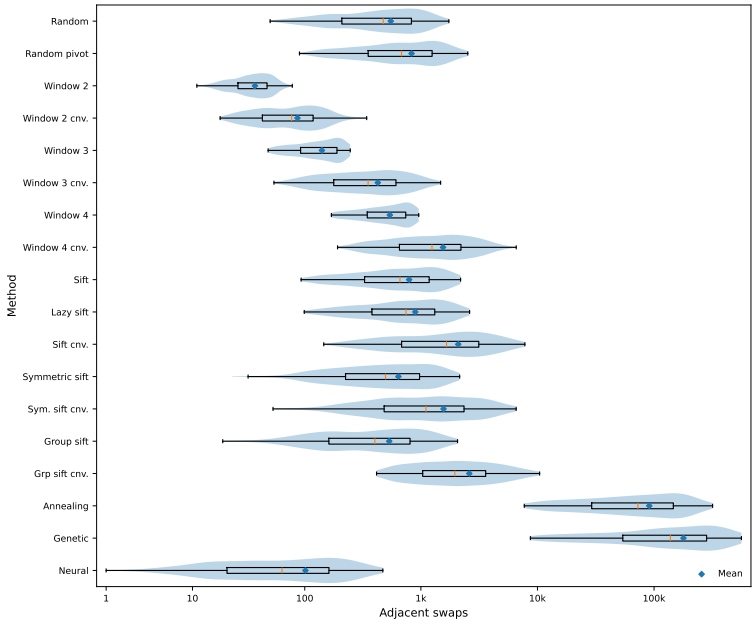


Figure 14: Distribution of adjacent-swap counts across the 500 Aralia fault-tree functions for the neural controller and all 17 heuristic baselines.

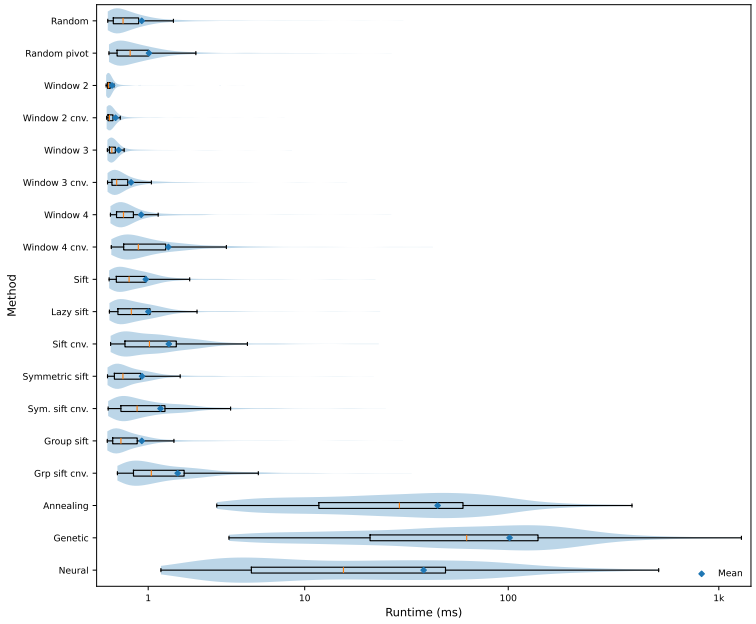


Figure 15: Distribution of reordering runtime across the 500 Aralia fault-tree functions for the neural controller and all 17 heuristic baselines.

D Additional Implementation Details

Experiments were conducted on a server equipped with an Intel Xeon w9-3595X CPU and an NVIDIA RTX PRO 6000 Blackwell Max-Q Workstation Edition GPU (96 GB).

D.1 Tabular RL

The tabular learner uses $\gamma = 1$, learning rate $\alpha = 1$, zero-initialized action values, and a fixed zero value for STOP. The behavior archive retains at most 16 endpoint orderings, enforcing a minimum spacing of $\lfloor n/2 \rfloor$ insertion moves between retained entries. Training runs for $20{,}000$ episodes with independent ϵ-greedy source and destination decisions, using geometric decay from $\epsilon = 1$ to 0.02. Archive guidance is available throughout training, but archive-derived values are not used directly as TD targets. Reverse completed-episode Bellman backups and intermediate slide-observation backups are enabled, while immediate reverse moves are blocked during exploration. After $10{,}000$ episodes, training additionally uses directed prefixes toward promising archived endpoints before resuming ordinary exploration; these prefix moves consume the normal episode budget. We use $\lambda_s = 5 \times 10^{-5}$ and $\lambda_m = 2 \times 10^{-5}$ and a move budget of $\lceil 1.5(n-1) \rceil$. The deliberately small penalty coefficients are chosen to minimize interference with the primary size-reduction objective, while still preferring shorter and fewer moves among otherwise similar solutions. Because the teacher is learned tabularly rather than through function approximation, these small reward differences can be represented directly without creating the optimization difficulties they might introduce in a neural value learner.

D.2 Neural Controller

Target generation. Each function contributes 50 initial orderings: the original ordering used for the full $20{,}000$-episode discovery run and 49 additional random starts, each receiving 500 refresh episodes while sharing the function's learned tables, archives, and ordering cache. Supervision is extracted from greedy trajectories, with up to four remaining-budget variants sampled per ordering. Four source variables are selected for dense destination supervision at each state–budget pair, and all $n - 1$ destinations of each selected source are labelled. Archive-derived continuation values are used where available, with short continuation rollouts used to fill otherwise unsupported targets. The resulting dataset contains 117,378,748 training and 12,473,312 validation state–budget examples from 51,409 and 5,396 functions, respectively, with ancestry groups kept within a single split.

Architecture. The final controller uses embedding dimension $d = 256$ and contains approximately 6.48M trainable parameters. Level positions are encoded by a shared $2 \to 64 \to 256$ MLP with GELU activation from normalized physical level and $\log n$. The bottom-up BDD encoder uses a shared $769 \to 256 \to 256 \to 256$ MLP with ReLU hidden activations. Within each level, a learned 256-dimensional query is updated through four shared cross-attention blocks with eight heads and 512-dimensional feed-forward layers, followed by LayerNorm. The resulting summary is concatenated with 55 structural features, projected from 311 to 256 dimensions, and combined with the level embedding. Inter-level reasoning uses four pre-normalized Transformer layers with eight attention heads and 1,024-dimensional feed-forward layers. The source and destination heads use shared one-hidden-layer MLPs with 512 hidden units and GELU activations; STOP retains a fixed zero source logit. Attention and Transformer dropout are zero.

Structural features. Each BDD level is represented by 55 handcrafted structural features computed from the current diagram. These are grouped into five families: (1) *outgoing-edge and terminal structure* (8 features), describing high/low terminal-edge fractions, edge spans, and skipped levels; (2) *incoming connectivity and sharing structure* (23), describing node indegrees, shared children, incoming-edge spans, complemented edges, parent-level diversity and entropy, and high/low child sharing; (3) *reach-weighted structure* (7), describing the probability of reaching a level and how indegree, complementation, and edge spans are distributed over more frequently reached nodes; (4) *subdiagram depth and remaining decision structure* (8), describing node heights, expected remaining decision-node visits, and the fraction of variables reachable below each node, both uniformly and weighted by reach probability; and (5) *Boolean branch structure* (9), describing high- and low-branch truth probabilities, their signed and absolute differences, and the entropy of repre-

sented subfunctions. Features are normalized by diagram size or variable count where appropriate, with logarithmic scaling for degree-related quantities. No dataset-fitted standardization is applied. Full feature definitions are provided in the accompanying implementation.

Distillation and optimization. Policy distillation uses temperature $\tau = 10^{-4}$ and hard-loss weight $\lambda_h = 0.25$ for the source, destination, and joint hard-action terms. The deliberately small temperature sharpens the teacher distributions so that the small action-value differences induced by the intentionally tiny swap and move penalties used during tabular training remain influential during distillation. Exact teacher ties receive uniform target mass. Models are optimized with AdamW using weight decay 10^{-4}, default $(0.9, 0.999)$ betas, and global gradient-norm clipping at 1.0. Training uses float32 precision with an effective accumulation target of 512 state–budget examples per optimizer update. The learning rate is 3×10^{-4} for epochs 1–5, 10^{-4} for epoch 6, and 3×10^{-5} for epoch 7; optimizer state is retained across these stages. All reported neural evaluations use the epoch-6 checkpoint.

Inference and timing. Evaluation uses a greedy source–destination rollout with budget $\lceil 1.5(n-1) \rceil$ and immediate termination on STOP. Previously visited full-move endpoint orderings are masked to prevent cycling. The reported best-seen result is the smallest BDD encountered during the complete rollout, including intermediate adjacent-swap states. Both swap counts and runtime include the full executed trajectory and the subsequent restoration to the best-seen ordering. Native inference integrates CUDD traversal, structural-feature extraction, masking, action selection, neural inference, and variable swapping in C++/LibTorch, with GPU execution for the neural model and no batching across test functions. Timed reordering includes feature construction, transfers, neural inference, action masking, and CUDD swaps, while initial BDD construction, model loading, and one-time warm-up are excluded.

D.3 BENCHMARK FUNCTION EXTRACTION PROCEDURE

For each source collection, we first convert the original representation into a Boolean dependency graph. This is a directed acyclic graph in which leaves correspond to independent Boolean inputs, internal nodes compute Boolean operations, and edges represent functional dependencies between signals. A Boolean function can then be extracted by selecting a node as a root and tracing its dependencies backwards to a chosen boundary. The boundary signals are treated as the independent inputs of the extracted function, while the dependency graph between the boundary and the root specifies how those inputs are combined to produce the root value. For the circuit-derived datasets, we consider both output and internal signals as candidate roots and sample backward cuts whose boundaries yield functions in the desired support range. Boundary signals need not be primary inputs of the original circuit: once a cut is made, they are deliberately treated as independent variables. For the Aralia fault-tree evaluation, we use the same dependency-graph abstraction but do not introduce artificial internal boundaries. Instead, for each selected fault-tree gate we recursively follow its dependencies to the original basic events and treat those events as the function inputs. Each extracted function is subsequently compiled into a reduced ordered BDD using CUDD. We define its variable count using the actual Boolean support of the resulting function, since some syntactic boundary variables may disappear through Boolean simplification. We then apply lightweight structural filters to remove functions whose BDDs are so small or structurally simple that variable reordering offers little meaningful scope for optimization. We group collected functions according to the benchmark instances and source ancestry from which they were derived and preserve these groupings when constructing the training and validation splits. The held-out circuit evaluation set is additionally screened to exclude functions associated with recorded training ancestry. Because several benchmark collections contain overlapping or mirrored source material, we audited the collected functions for duplicates and removed them.