LEARNING COMPACT BOOLEAN NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Deploying machine learning in resource-constrained and latency-sensitive domains is challenging. Boolean networks offer a promising alternative to floating-point neural networks by using Boolean gates to substantially reduce cost and latency. However, learning compact and accurate Boolean networks remains hard because of their discrete, combinatorial structure. We address this through three novel, complementary contributions: (i) a parameter-free strategy for learning effective connections, (ii) a compact convolutional Boolean architecture that exploits spatial locality while requiring fewer Boolean operations than existing convolutional kernels, and (iii) an adaptive discretization procedure that reduces the accuracy drop when converting a continuously relaxed network into a discrete Boolean network. Across standard vision benchmarks, our method improves the Pareto frontier over prior state-of-the-art methods, achieving higher accuracy with up to $47\times$ fewer Boolean operations. This advantage extends to other modalities. On an FPGA, our model on MNIST achieves 99.38% accuracy with a latency of 6.48 ns, surpassing the prior state of the art in both accuracy and runtime while producing a $7\times$ smaller Verilog file.

1 INTRODUCTION

Machine learning has found applications across a wide range of domains, with neural networks as the dominant modeling approach. Their high inference costs and substantial computational and memory requirements, however, prohibit deployment in resource-constrained and latency-sensitive settings. For example, wearable devices and battery-powered IoT sensors often need to process data locally and respond in real time; at the Large Hadron Collider, real-time event selection requires latencies in the order of nanoseconds (Duarte et al., 2019). In these applications, memory usage, inference latency, and energy consumption are the primary concerns. A promising solution is to use Boolean networks, which operate directly on Boolean values, i.e., 0 and 1, instead of on floating-point numbers, because Boolean operations are inherently more efficient in terms of computation, memory, and power consumption.

A smaller Boolean circuit is generally cheaper to execute. For a given Boolean function, finding a compact Boolean circuit implementing it is a fundamental and well-studied problem (Rudell & Sangiovanni-Vincentelli, 1987; Ganai & Kuehlmann, 2000; De Micheli, 1994; Bjesse & Boralv, 2004; Mishchenko et al., 2006; Riener et al., 2022). However, in practice, the Boolean function is often not known a priori and must be learned from data.

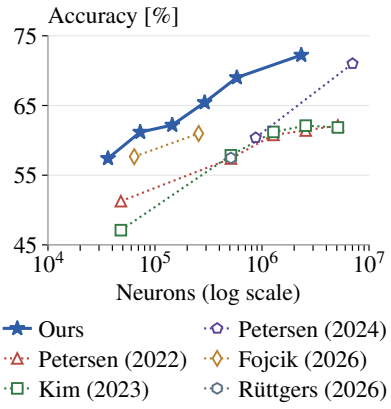


Fig. 1: Test accuracy versus neuron count of our method and prior work on Boolean networks on CIFAR-10.

This Work: Learning Compact and Accurate Boolean Networks Efficiently. We build on the learning framework proposed by Petersen et al. (2022). It works by first relaxing a Boolean network into a differentiable representation and training it with gradient-based methods, and then discretizing it into a Boolean network. Despite existing efforts, three key challenges remain open. (i) *Efficient connection learning*. Existing methods either use randomly sampled, fixed connections (Petersen et al., 2022; 2024; Rüttgers et al., 2026) (i.e., without connection learning) or parameterize connections with ad-

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.

ditional weight matrices (Bacellar et al., 2024; Fojcik et al., 2026). By the lottery ticket hypothesis (Frankle & Carbin, 2019), the extreme sparsity of Boolean networks (two inputs per neuron) makes random connections highly ineffective. The weight matrix parameterization, on the other hand, induces significant parameter overhead during training, typically quadratic in the number of neurons. How to learn effective connections efficiently remains an open question. (ii) *Compact convolutional architecture*. The only existing convolutional Boolean architecture (Petersen et al., 2024) uses a hardcoded tree-like kernel, which requires more Boolean operations (exponential in the depth of the tree) than a feedforward layer. (iii) *Discretization-aware training*. The discrepancy between the relaxed and discrete networks causes a clear accuracy drop after discretization.

In this paper, we address these three challenges of learning compact and accurate Boolean networks. Our main contributions are as follows.

- We introduce a parameter-free connection-learning strategy. By adaptively resampling candidate input-function triples, our method can learn effective sparse connectivity while avoiding the heavy matrix parameterization used in prior work.
- We design a compact convolutional architecture that exploits spatial locality with single-operation Boolean kernels, substantially reducing the size of convolutional Boolean networks compared to prior work.
- We develop an adaptive discretization strategy that progressively discretizes layers during training, narrowing the discretization gap and improving the performance of the final discrete model.
- Extensive evaluations show that our method substantially improves the Pareto frontier (Fig. 1), achieving higher accuracy with up to $47\times$ fewer Boolean operations. Additional results on tabular and sequence data, as well as FPGA synthesis on an edge device, further demonstrate the generality and inference-time efficiency of our approach.

2 Related Work

Improving Inference Efficiency. Many works reduce the inference cost of floating-point neural networks through quantization or pruning. Quantization reduces computation by lowering the precision of weights and activations (Umuroglu et al., 2017; Zhan et al., 2021; Ngadiuba et al., 2021; Ji et al., 2023; Plagwitz et al., 2023), while pruning reduces computation by removing connections or units (Posewsky & Ziener, 2018; Duarte et al., 2019). Boolean networks push both directions to the extreme: they operate directly at the Boolean level, using only 16 fixed neuron types with two inputs per neuron. Unlike many quantization or pruning methods, this design generally requires tailored training from scratch rather than direct conversion of a trained floating-point network. Fig. 2 compares the accuracy and latency of representative quantization and pruning methods with those of our Boolean networks on MNIST, highlighting a fundamentally different tradeoff.

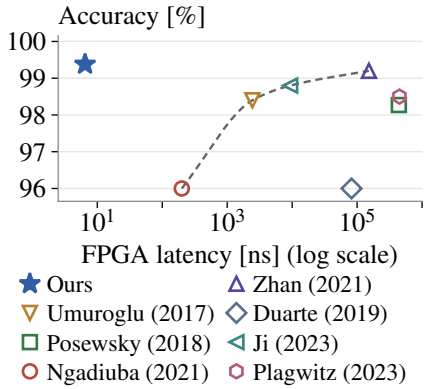


Fig. 2: Comparison between quantized or pruned networks and our Boolean networks. Dashed curve is a smooth interpolation of the displayed quantization/pruning Pareto points.

Learning Boolean Structures. We review three families of learned Boolean structures: Boolean networks, binary decision diagrams (BDDs), and lookup tables (LUTs). Petersen et al. (2022) introduced the differentiable learning framework for Boolean networks, followed by efforts in reducing training overhead (Rüttgers et al., 2026), learning inter-layer connections (Bacellar et al., 2024; Fojcik et al., 2026), convolutional design (Petersen et al., 2024), and recurrent architectures (Bührer et al., 2025). For BDDs, exact learners formulate structure learning with SAT/MaxSAT (Hu et al., 2022; Shati et al., 2023) or mixed-integer programming (Florio et al., 2023). Zantedeschi et al. (2021) train discrete decision trees via argmin differentiation, and Qiu et al. (2025) parameterize ordered BDDs with continuous encodings before extracting discrete BDDs. For LUTs, prior work reduces sparse quantized networks into LUT networks (Umuroglu et al., 2020), trains LUT networks via Lagrange-interpolating polynomial relaxations (Wang et al., 2019), incorporates hardware-aware training and assembly (Andronic & Constantinides, 2025), performs post-training LUT compression

(Cassidy et al., 2025), and assembles small LUT subnetworks into larger LUT networks (Weng et al., 2025).

3 BACKGROUND

Boolean Circuits and Boolean Networks. Boolean circuits are directed acyclic graphs in which each computational node evaluates a bivariate Boolean function $B : \{0,1\}^2 \to \{0,1\}$. We denote the 16 possible bivariate Boolean functions by $B_i : \{0,1\}^2 \to \{0,1\}, i = 1, \ldots, 16$ (see Appendix F). A Boolean network is a Boolean circuit whose computational nodes are arranged in layers, with connections between successive layers. Fig. 3 illustrates both structures.

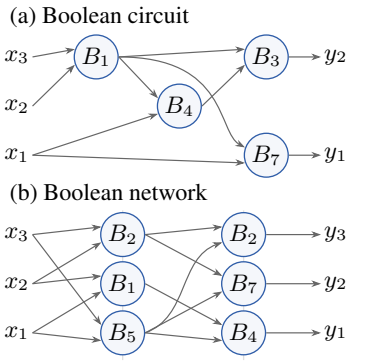


Fig. 3: A Boolean circuit (a) and a layered Boolean network (b). Each gate has two inputs.

Boolean circuits realize only Boolean functions. To represent a non-Boolean function $g : A \to B$, we combine a Boolean function $f : \{0,1\}^m \to \{0,1\}^n, m, n \in \mathbb{N}$, with an encoder $\mathcal{E} : A \to \{0,1\}^m$ and a decoder $\mathcal{D} : \{0,1\}^n \to B$, such that $g = \mathcal{D} \circ f \circ \mathcal{E}$. In particular, for an image classification function $g : [0,1]^d \to \mathcal{C}$ that maps a real-valued image to a class label, we use the thermometer encoder $\mathcal{E}(\boldsymbol{x}) := [\mathbb{1}\{\boldsymbol{x} \geq \frac{1}{N}\}, \ldots, \mathbb{1}\{\boldsymbol{x} \geq \frac{N-1}{N}\}]$, where $\mathbb{1}\{\cdot\}$ is the indicator function applied elementwise and N controls the encoding resolution. The population-count decoder (also called GroupSum) selects the class with the most 1s in the output Boolean array $\boldsymbol{Y} \in \{0,1\}^{|\mathcal{C}| \times n_c}$, with n_c output bits per class. It is defined as $\mathcal{D}(\boldsymbol{Y}) := \arg\max_{c \in \mathcal{C}} \sum_{i=1}^{n_c} \boldsymbol{Y}_{c,i}$.

Learning Boolean Networks via Differentiable Relaxations. To learn a Boolean network from data, Petersen et al. (2022) propose to relax discrete Boolean networks into differentiable functions, enabling training via gradient descent. Specifically, for $i = 1, \ldots, 16$, let $B_i^c : [0,1]^2 \to [0,1]$ be a differentiable relaxation of B_i. For example, $B_2(x_b, y_b) = x_b \wedge y_b, x_b, y_b \in \{0,1\}$, is relaxed into $B_2^c(x,y) = xy, x, y \in [0,1]$, satisfying $B_2^c(x_b, y_b) = B_2(x_b, y_b)$. Each neuron is then parameterized as a weighted sum of the relaxed functions:

$$f^c(x,y) = \sum_{i=1}^{16} \frac{\exp(\boldsymbol{w}_i)}{\sum_{j=1}^{16} \exp(\boldsymbol{w}_j)} B_i^c(x,y), \tag{1}$$

where $\boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{16})$ is the trainable parameter vector for each neuron. The relaxed network is trained by updating these vectors. Finally, each neuron is discretized back to a bivariate Boolean function by

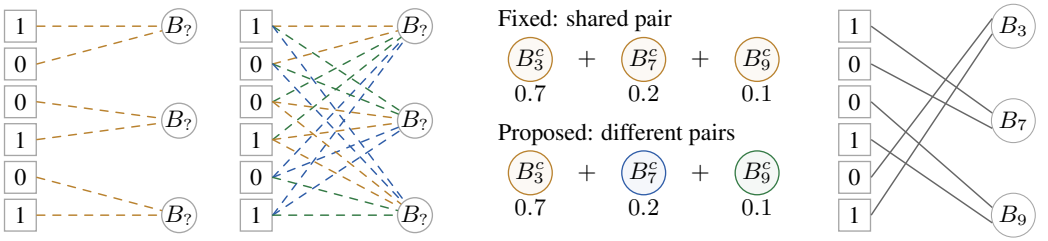
$$f := B_{i^*}, \quad \text{with } i^* = \arg\max_i \boldsymbol{w}_i. \tag{2}$$

This relaxation learns the Boolean operation at each neuron while keeping the network connections fixed. We address efficient connection learning in §4.

Convolutional Boolean Networks. Convolutional layers capture local spatial structure in floating-point neural networks. Petersen et al. (2024) design a convolutional architecture for Boolean networks in which each kernel is a complete binary tree of Boolean operations. A tree of depth d requires $2^d - 1$ Boolean operations to combine 2^d inputs into one output. This exceeds the single operation per neuron in a non-convolutional Boolean layer when $d > 1$.

4 LEARNING COMPACT BOOLEAN NETWORKS

We introduce our methods for learning compact and accurate Boolean networks. We first present an efficient connection learning strategy in §4.1, based on a novel Boolean-neuron parameterization that aggregates information from multiple connections and an adaptive resampling strategy that continuously explores new connections. Next, we present a compact convolutional Boolean architecture in §4.2, replacing tree structures with single-operation kernels enabled by connection learning. Finally, we introduce an adaptive discretization strategy in §4.3 that progressively discretizes network layers during training to reduce the resulting accuracy drop.



(a) Fixed input pair (b) Learnable pairs (c) Weighted aggregation (d) Discrete neurons

Fig. 4: Fixed connections (a) and our learnable connections (b). Each relaxed neuron aggregates candidate Boolean operations on a shared input pair or on different pairs, respectively (c, three candidates shown). Discretization retains one operation and two inputs per neuron (d). $B_?$ denotes an undecided Boolean function.

4.1 Efficient Connection Learning

We modify the parameterization in Equation (1) to learn connections. Instead of parameterizing neurons as a weighted aggregation of sixteen Boolean operations with the same inputs, we assign different inputs to different Boolean operations. Specifically, let $\boldsymbol{x} \in [0,1]^d$ represent the activations of the d neurons in the ℓ-th layer. We parameterize a neuron in the $(\ell+1)$-th layer as

$$\tilde{f}^c(\boldsymbol{x}; \boldsymbol{k}, \boldsymbol{p}, \boldsymbol{q}) = \sum_{i=1}^{16} \frac{\exp(\boldsymbol{w}_i)}{\sum_{j=1}^{16} \exp(\boldsymbol{w}_j)} B^c_{\boldsymbol{k}_i}(\boldsymbol{x}_{\boldsymbol{p}_i}, \boldsymbol{x}_{\boldsymbol{q}_i}), \tag{3}$$

where $\boldsymbol{k} \in \{1, \ldots, 16\}^{16}$ are indices denoting the Boolean operations, and $\boldsymbol{p}, \boldsymbol{q} \in \{1, \ldots, d\}^{16}$ index the two input neurons for each candidate operation in the ℓ-th layer. At the beginning of training, $(\boldsymbol{k}, \boldsymbol{p}, \boldsymbol{q})$ can be sampled uniformly at random or initialized with specific assignments. Fig. 4 illustrates the parameterization.

For each neuron, the optimal combination of inputs and Boolean function might not be included in the initial assignment. To learn better combinations, we propose an adaptive resampling strategy. Once a neuron becomes "stable" during training, we resample its $(\boldsymbol{k}, \boldsymbol{p}, \boldsymbol{q})$ candidates to explore new combinations. We measure the stability of a neuron by monitoring its weight entropy, defined as

$$h(\boldsymbol{w}) = -\sum_{i=1}^{16} \tilde{\boldsymbol{w}}_i \log \tilde{\boldsymbol{w}}_i, \text{ where } \tilde{\boldsymbol{w}}_i = \frac{\exp(\boldsymbol{w}_i)}{\sum_{j=1}^{16} \exp(\boldsymbol{w}_j)}.$$

The weight entropy is zero if and only if the normalized weight vector $\tilde{\boldsymbol{w}}$ is one-hot, so the neuron is represented by a single $(\boldsymbol{k}_i, \boldsymbol{p}_i, \boldsymbol{q}_i)$ triple. It reaches its maximum when all elements are equal, corresponding to an equal mixture of all candidate triples. We track the exponential moving average (EMA) of the weight entropy for each neuron individually. When the EMA of $h(\boldsymbol{w})$ for a neuron converges, its weight vector $\boldsymbol{w}$ can have three possible states: (i) dominated by a single triple $(\boldsymbol{k}_i, \boldsymbol{p}_i, \boldsymbol{q}_i)$, i.e., $\exists i^*, \tilde{\boldsymbol{w}}_{i^*} \geq 0.95$, (ii) dispersed, i.e., $\max_i \tilde{\boldsymbol{w}}_i \leq 0.4$, or (iii) concentrated without a dominant triple. For (i), further training of the neuron is unlikely to improve the discretized network unless the dominant triple is replaced, so we resample all $(\boldsymbol{k}_i, \boldsymbol{p}_i, \boldsymbol{q}_i)$ except the dominant one to explore new combinations. For (ii), the neuron has not found a useful triple, so we resample all $(\boldsymbol{k}_i, \boldsymbol{p}_i, \boldsymbol{q}_i)$ to restart its learning. Neurons in case (iii) are left unchanged since they are still learning. Full pseudocode for the resampling strategy is provided in Algorithm 1 in Appendix A. After training, most neurons (frequently over 90%) empirically fall into case (i), indicating that they have learned meaningful patterns expressed by a single dominant input pair and Boolean function.

We remark on the relation to prior works and the overhead of the proposed method. As mentioned in §1, Petersen et al. (2022; 2024) and Rüttgers et al. (2026) randomly sample connections and keep them fixed during training. To learn connections, Bacellar et al. (2024) and Fojcik et al. (2026) propose to parameterize the connections with additional weight matrices. A Boolean layer with d_{in} input neurons and d_{out} output neurons additionally maintains a weight matrix $W_{\text{link}} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$. However, such large matrices are often impractical: for a typical medium-sized model in Petersen et al. (2024), W_{link} for a single layer can consume over 3400 GB of memory. In contrast, our method learns network connections without additional parameters.

4.2 Convolution without Boolean Trees

Convolution is a key component in floating-point neural networks to capture spatial features, especially for vision tasks. A simple way to construct convolutional layers in Boolean networks is to take a single Boolean operation as the convolutional kernel. However, such kernels under fixed connections only observe two inputs from the receptive field. With the design in §4.1, a convolutional kernel learns to extract important information from the full receptive field by aggregating multiple input pairs during training. Each kernel can thus observe up to 32 distinct inputs from the receptive field. As a result, we can learn a compact convolutional Boolean network based on single-operation kernels.

In contrast, the convolutional kernels in Petersen et al. (2024) are built on Boolean trees. Specifically, 2^d neurons sampled from the receptive field are fed into a binary Boolean tree of depth d, and each internal node is learned with the relaxation in Equation (1). This tree structure has two main issues. First, the number of Boolean operations per layer grows exponentially with the tree depth d, leading to high inference costs. Second, the hierarchical structure of binary trees introduces sequential dependencies that limit parallelism during both forward and backward passes. As we later demonstrate empirically in §5, our method can learn much smaller convolutional Boolean networks with higher accuracies.

4.3 Adaptive Discretization

As introduced in §3, after training, the network is discretized to Boolean by replacing all relaxed neurons with their dominant Boolean operation at once. This training paradigm does not account for the discrepancy between the continuous and Boolean networks, leading to a systematic accuracy drop after discretization.

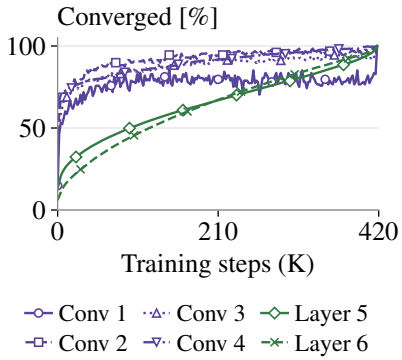


Fig. 5: Convergence speed of different layers.

During training, convolutional layers of the network tend to converge faster than the non-convolutional layers. We measure the fraction of neurons that have the same dominant $(\boldsymbol{k}_i, \boldsymbol{p}_i, \boldsymbol{q}_i)$ triple as the final discretized network. As shown in Fig. 5, convolutional layers converge much faster than non-convolutional layers. Further, convolutional layers tend to be less stable, e.g., the first convolutional layer stops improving when roughly 80% of neurons converge, while non-convolutional layers converge more steadily. Based on these observations, we progressively discretize and freeze layers from shallow to deep during training. Specifically, we monitor the convergence of each layer using the average weight entropy and EMAs, similar to §4.1. Once the first layer converges, we discretize it using Equation (2), freeze it, and continue training the subsequent layers. We proceed layer-wise in this manner for all convolutional layers, and then continue training the non-convolutional layers in their relaxed form until the end. Full pseudocode for the adaptive discretization strategy is provided in Algorithm 2 in Appendix A. Progressive discretization forces the layers to adapt to Boolean inputs during training. This reduces the accuracy drop caused by discretization.

We remark that Kim (2023) and Yousefi et al. (2025) also attempt to mitigate discretization error by injecting noise during training. However, the injected noise distribution does not accurately reflect the true distribution of the discretization error, and thus fails to guide the training towards better solutions (see §5.3).

5 Experiments

We evaluate the proposed components both in isolation and in combination. Details on input encoding, data augmentation, baselines, architectures, and training hyperparameters are provided in Appendix H. We first evaluate learned connections (§5.1), compact convolutions (§5.2), and adaptive discretization (§5.3) in isolation, then combine all components to evaluate the final Pareto frontier and FPGA inference performance (§5.4). We compare training costs in §5.5.

Table 1: Connection learning on CIFAR-10.

Method	Acc.	*#neurons*	*#BOPs*
DiffLogicNet-small	52.84%	48.0 K	27.0 K
DiffLogicNet-medium	59.90%	512 K	312 K
DiffLogicNet-large	62.57%	1.28 M	780 K
Ours-small	56.13%	48.0 K	27.0 K
Ours-medium	62.84%	512 K	297 K
Ours-large	64.53%	1.28 M	705 K

Table 2: Boolean convolution on CIFAR-10.

Method	Acc.	*#neurons*	*#BOPs*
TreeLogicNet-S	59.79%	874 K	521 K
TreeLogicNet-M	71.37%	7.00 M	3.66 M
Ours-S	65.56%	291 K	161 K
Ours-M	68.73%	582 K	307 K
Ours-L	72.09%	2.33 M	1.09 M

Metrics. We report test accuracy as the prediction-performance metric. To measure Boolean circuit size before hardware synthesis, we report both the number of neurons in the Boolean network (*#neurons*) and the number of Boolean operations after simple pruning (*#BOPs*). This pruning removes redundant neurons that do not contribute to the final output, as well as Identity and NOT neurons by rewiring their input connections to their output connections. See Appendix G for details. We apply the same pruning procedure to all methods to avoid bias toward any particular circuit optimization method. For FPGA inference, we generate Verilog from the pruned Boolean networks and estimate runtime per sample using the heuristic timing model described in Appendix J.

5.1 Connection Learning

We evaluate the connection-learning method in §4.1 against DiffLogicNet (Petersen et al., 2022) under matched architecture and data processing settings, differing only in the neuron parameterization.

Main Results. Table 1 summarizes the results on CIFAR-10. Across all model sizes, our learned connections consistently outperform DiffLogicNet's randomly sampled fixed connections. Notably, our 512K-neuron model is more accurate than a DiffLogicNet model with 1.28M neurons. On average, connection learning improves CIFAR-10 accuracy by 2.73%. Results on MNIST are consistent, as reported in Appendix D.

Learning More Layers Helps. Bacellar et al. (2024) reported that, for their connection-learning algorithm, learning only the first-layer connections can match fully learnable models, suggesting limited effectiveness of connection learning in deeper layers. We test whether our method learns useful connections throughout the network by varying the number of learned-connection layers in the small CIFAR-10 model, while keeping fixed random connections in the remaining layers as in DiffLogicNet. As shown in Fig. 6, accuracy consistently improves as more layers learn connections, indicating that our method is effective in both shallow and deep layers.

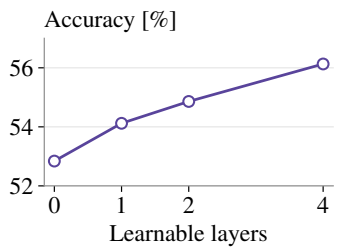


Fig. 6: Effect of the number of learned-connection layers on CIFAR-10.

Resampling vs More Candidates. As discussed in §4.1, resampling explores new candidate triples during training. A direct alternative is to increase the number of candidate triples $(\boldsymbol{k}_i, \boldsymbol{p}_i, \boldsymbol{q}_i)$ per neuron from 16 to a larger value K, allowing each neuron to aggregate information from a larger pool. As shown in Fig. 7, increasing K from 16 to 128 significantly improves accuracy when resampling is disabled, from 53.24% to 54.94%. With resampling enabled, however, the gain is marginal, from 56.14% to 56.48%. Further, resampling consistently outperforms the non-resampling counterpart for all K. Thus, resampling effectively learns good connections, making the parameter overhead of maintaining many candidates per neuron unnecessary.

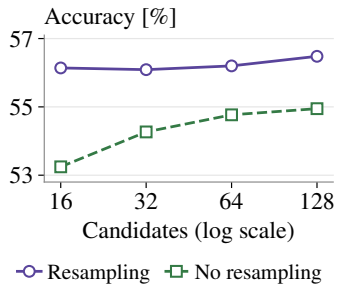


Fig. 7: Effect of candidate count with and without resampling.

Hyperparameters. We fix the hyperparameters involved in the connection-learning technique, including the "dominant" and "dispersed" state thresholds, the patience parameter, and the EMA momentum, globally in all experiments in §5 without model- or dataset-specific tuning. The ablation

study in Appendix D.4 shows that test accuracy is stable across reasonable ranges of these hyperparameters.

5.2 Single-Operation Boolean Convolution

We evaluate the proposed convolutional architecture, which uses a single Boolean operation as the convolutional kernel, against TreeLogicNet (Petersen et al., 2024), which uses Boolean-tree kernels. Both architectures use 3×3 receptive fields. TreeLogicNet restricts each kernel to two neighboring channels, while our final models restrict each kernel to one channel. We show that this restriction improves performance, and we hypothesize that this effect arises from thermometer encoding on natural images.

Main Results. Table 2 summarizes the results on CIFAR-10. Our architecture consistently outperforms TreeLogicNet with fewer neurons: our large model achieves 0.72% higher accuracy (72.09% vs. 71.37%) with 3x fewer neurons (2.33M vs. 7M). We exclude the larger models of Petersen et al. (2024) because they include floating-point feature-extraction layers and are therefore not Boolean networks. Results on MNIST show consistent improvements, as reported in detail in Appendix D.

The Paradox of Channel Visibility. To isolate the contribution of the convolutional architecture from channel visibility, we compare our method (582K model) and TreeLogicNet (874K model) under matched channel-visibility conditions. As shown in Fig. 8, our smaller model consistently outperforms TreeLogicNet across all channel visibilities. This supports the advantage of connection-learned single-operation kernels over hard-coded Boolean-tree kernels. Since TreeLogicNet uses a hard-coded Boolean-tree kernel, applying our connection-learning strategy to it would require a substantial redesign. We therefore cannot further match connection learning across these two architectures.

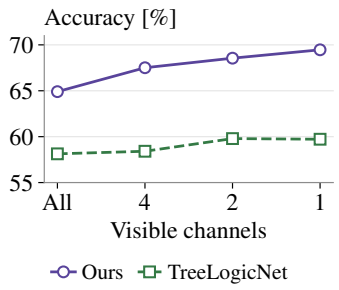


Fig. 8: Comparison with TreeLogicNet under matched channel-visibility conditions.

This study reveals a paradox of channel visibility. Consistent with Petersen et al. (2024), limiting the number of accessible channels substantially improves performance on CIFAR-10. In particular, restricting each kernel to one channel achieves the best accuracy, 4.56% higher than using all channels (69.46% vs. 64.90%).

This is surprising because one-channel visibility prevents convolutional layers from mixing information across channels. We hypothesize that the effect arises from thermometer encoding on natural images. As visualized in Appendix B.1, thermometer encoding can break complementary cross-channel information and sometimes removes information from some channels almost entirely. This may cause interference when a kernel aggregates multiple channels. Additional experiments in Appendix B.2 consistently support the paradox and this hypothesis. We leave a more extensive study and the search for a solution to future work.

5.3 Adaptive Discretization

We evaluate adaptive discretization against Gumbel noise injection on both final test accuracy and the discretization gap, which is defined as the accuracy difference between the relaxed model before discretization and the discrete Boolean network after discretization, following Yousefi et al. (2025).

Main Results. Fig. 9 compares adaptive discretization with Gumbel noise injection on CIFAR-10 using our medium convolutional model. We sweep the Gumbel-noise hyperparameter for a fair comparison. Adaptive discretization consistently improves test accuracy while reducing training cost: after four convolutional layers are discretized, training is 1.8x faster than the fully relaxed model, since discretized layers are cheaper in both forward and backward passes. By contrast, Gumbel noise injection adds computational overhead

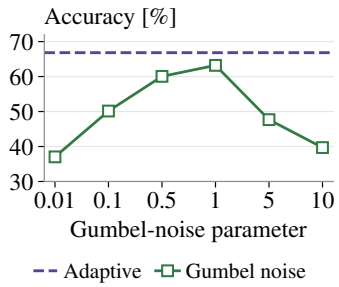


Fig. 9: Gumbel noise injection versus adaptive discretization. Parameter values are equally spaced for display.

Table 3: Final-model comparison with TreeLogicNet.

Dataset	Method	Acc. (%)	*#neurons*	*#BOPs*	Verilog file size (MB)	FPGA latency (ns)
MNIST	TreeLogicNet-S	97.77	185 K	117 K	11.24	6.96
	TreeLogicNet-M	98.96	740 K	427 K	34.62	7.74
	TreeLogicNet-L	99.33	11.8 M	6.20 M	484.59	8.76
	Ours-S	99.31	260 K	130 K	32.42	6.18
	Ours-M	99.38	520 K	251 K	63.60	6.48
CIFAR-10	TreeLogicNet-S	59.79	874 K	521 K	49.27	8.16
	TreeLogicNet-M	71.37	7.00 M	3.66 M	337.52	9.06
	Ours-T	62.71	145 K	85.2 K	19.67	5.88
	Ours-S	67.12	291 K	163 K	38.34	6.18
	Ours-M	70.22	582 K	319 K	76.41	6.48
	Ours-L	72.56	2.33 M	1.08 M	266.93	7.08

throughout training. In addition, ablation studies in Appendix D.2 show that well-tuned fixed-schedule discretization could also match the performance of adaptive discretization, highlighting the importance of discretization-aware training. Nevertheless, adaptive discretization is more convenient in practice because it does not require tuning additional discretization schedules.

Progressive Discretization Reduces the Gap. We further vary the number of layers discretized during training from 1 to 5 in the medium convolutional CIFAR-10 model, which has 6 layers in total. As shown in Fig. 10, discretizing more layers progressively reduces the discretization gap, confirming the intended effect of adaptive discretization. Discretizing all layers during training does not yield the best final accuracy under the fixed training budget, likely because early discretization limits the model's remaining learning capacity. In practice, we adaptively discretize all convolutional layers across all settings, balancing gap reduction with sufficient optimization, since non-convolutional layers already converge steadily without training-time discretization (see Fig. 5).

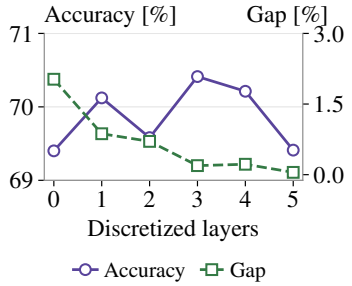


Fig. 10: Accuracy (circles, left) and discretization gap (squares, right) versus discretized layers.

5.4 Combined Results and FPGA Inference

We combine all proposed components and evaluate the final models on MNIST and CIFAR-10. Table 3 summarizes accuracy, Boolean-network complexity, generated Verilog file size, and estimated FPGA inference latency. On MNIST, our small and medium models achieve 99.31% and 99.38% accuracy with 45x and 22x fewer neurons than TreeLogicNet-L (260K and 520K vs. 11.8M), respectively. The medium model also exceeds TreeLogicNet-L's accuracy. After pruning, the two models use 47x and 24x fewer Boolean operations (130K and 251K vs. 6.2M). On CIFAR-10, our large model reaches 72.56% accuracy, outperforming TreeLogicNet with 3x fewer neurons and Boolean operations.

We compile our models and the TreeLogicNet baselines to Verilog and estimate inference latency using the heuristic timing model detailed in Appendix J. Table 3 reports the generated Verilog file sizes, which quantify the size of the compiled source representation, and estimated FPGA latencies per input. On MNIST, our medium model improves accuracy over TreeLogicNet-L while reducing Verilog file size from 484.59 to 63.60 MB and estimated latency from 8.76 to 6.48 ns, a 26% reduction. On CIFAR-10, our large model improves accuracy over TreeLogicNet-M with a 21% smaller Verilog file (266.93 vs. 337.52 MB) and 22% lower estimated latency (7.08 vs. 9.06 ns). These comparisons show improved accuracy alongside smaller compiled representations and lower estimated inference latency. We further analyze the gate distribution of these models in Appendix I.

To assess applicability beyond image classification, we also evaluate our method on image segmentation, tabular classification, and sequence classification. The competitive results reported in Appendix D.3 support its applicability across tasks and data modalities.

Table 4: Training cost on CIFAR-10.

Method	Throughput (it/s)	Time (h)	Acc. (%)
DiffLogicNet-small	417.7	0.20	52.84
DiffLogicNet-medium	207.1	0.40	59.90
DiffLogicNet-large	101.8	0.82	62.57
Ours-small	720.7	0.12	56.13
Ours-medium	65.6	1.27	62.84
Ours-large	20.1	4.15	64.53

Table 5: Training cost on CIFAR-10.

Method	Throughput (it/s)	Time (h)	Acc. (%)
TreeLogicNet-S	19.7	5.64	59.79
TreeLogicNet-M	10.2	10.84	71.37
Ours-T	41.6	2.67	62.71
Ours-M	10.5	10.58	70.22

5.5 Training Cost

We compare our training throughput and training time with those of DiffLogicNet and TreeLogicNet on CIFAR-10. For DiffLogicNet, we evaluate three matched architectures on a single RTX Pro 6000 GPU with batch size 128. We use three warm-up batches and average throughput over the subsequent 100 batches. Table 4 reports throughput, training time, and test accuracy. For the small architecture, our method is both faster and more accurate: training time decreases from 0.20 to 0.12 hours, while test accuracy increases by 3.29 percentage points. For the medium and large architectures, our method requires approximately $3.2\times$ and $5.1\times$ as much training time, respectively. This slowdown primarily reflects differences in the parameterizations used to learn network connections. The optimized DiffLogicNet implementation uses basis projection, an optimization reported to provide up to a $3.96\times$ training speedup on MNIST over full gate evaluation (Fojcik et al., 2026, Table 1). This optimization does not directly extend to our parameterization because it assumes that all candidate operations share the same input pair.

For the TreeLogicNet comparison, we use the same throughput measurement protocol on a single RTX 2080 Ti GPU with batch size 128. As shown in Table 5, our tiny model achieves substantially higher throughput than TreeLogicNet-S (41.6 vs. 19.7 it/s). For the medium models, our method achieves comparable throughput to TreeLogicNet-M (10.5 vs. 10.2 it/s). We attribute this competitive training efficiency to our simplified convolutional design, which helps offset the overhead of our connection-learning parameterization. We leave further training-time optimization to future work.

6 Discussion

Despite these promising results, several limitations remain. First, although our method reduces the number of neurons during training and the number of Boolean operations at inference, training still relies on floating-point computations. As each neuron maintains 16 floating-point weights, the training cost is substantially higher than the inference cost. Second, our training method does not incorporate advanced logic synthesis techniques. Using insights from these techniques to train synthesis-friendly networks may yield more compact circuits after synthesis. Finally, our method focuses on learning Boolean networks. The design of input encoding and output decoding schemes remains an important direction for future work.

7 Conclusion

This work presents a unified approach for learning compact and accurate Boolean networks. We address three bottlenecks in existing methods: ineffective fixed random connections, expensive Boolean-tree convolutional kernels, and the discretization gap between relaxed and discrete networks. Our method learns sparse connections without additional parameterization matrices, replaces Boolean-tree kernels with single-operation convolutional kernels, and progressively discretizes layers during training. Across vision benchmarks, these designs improve accuracy while substantially reducing Boolean-network complexity, achieving up to $47\times$ fewer Boolean operations than prior state-of-the-art methods. FPGA timing estimates indicate lower inference latency for the resulting networks, while additional experiments on image segmentation and other modalities support the broader applicability of the proposed approach.

BROADER IMPACT

This work aims to improve neural-network inference efficiency by learning compact Boolean networks that execute using simple Boolean operations. Potential benefits include lower inference latency, reduced memory and energy consumption, and easier deployment on resource-constrained hardware such as edge devices and FPGAs. These improvements may broaden access to efficient machine learning systems and reduce the environmental cost of large-scale inference.

The same efficiency gains may also make harmful or inadequately audited applications easier to deploy. Because Boolean networks are learned from data, they remain susceptible to biases and failure modes arising from their training data. Application-specific evaluations of safety, privacy, fairness, and robustness therefore remain essential before deployment.

REPRODUCIBILITY STATEMENT

We provide an anonymous codebase as a supplementary attachment submitted with this paper. Its `README.md` provides setup and usage instructions, and `experiments/configs` contains the configurations and commands to reproduce all experiments reported in the paper.

LLM DISCLOSURE

This paper was assisted by PIRA (PIRA Project, 2026), a research-assistant system powered by GPT-6-Astra. The assistance included partial implementation assistance and writing polish. The authors are fully responsible for the presented final content.

REFERENCES

Marta Andronic and George A. Constantinides. NeuraLUT-Assemble: Hardware-aware assembling of sub-neural networks for efficient LUT inference. In *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 208–216. IEEE, 2025. doi: 10.1109/FCCM62733.2025.00077.

Alan Tendler Leibel Bacellar, Zachary Susskind, Mauricio Breternitz, Jr., Eugene John, Lizy Kurian John, Priscila Machado Vieira Lima, and Felipe M.G. França. Differentiable weightless neural networks. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 2277–2295. PMLR, 21–27 Jul 2024.

P. Bjesse and A. Boralv. DAG-aware circuit compression for formal verification. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 42–49, 2004. doi: 10.1109/ICCAD.2004.1382541.

Simon Bührer, Andreas Plesner, Till Aczel, and Roger Wattenhofer. Recurrent deep differentiable logic gate networks. In *Proceedings of the 2nd International Workshop on Edge and Mobile Foundation Models*, pp. 31–36. ACM, 2025.

Oliver Cassidy, Marta Andronic, Samuel Coward, and George A. Constantinides. ReducedLUT: Table decomposition with "don't care" conditions. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 36–42. ACM, 2025. doi: 10.1145/3706628.3708823.

Giovanni De Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994.

Javier Duarte, Christian Herwig, Burt Holzman, Sergo Jindariani, Benjamin Kreis, Mia Liu, Ryan Rivera, Nhan Tran, Song Han, Phil Harris, Dylan Rankin, Vladimir Loncar, Jennifer Ngadiuba, Maurizio Pierini, Sioni Summers, Scott Hauck, Shih-Chieh Hsu, Zhenbin Wu, and Edward Kreinar. Low-latency machine learning inference on FPGAs. In *NeurIPS Workshop on Machine Learning and the Physical Sciences*, 2019.

Alexandre M. Florio, Pedro Martins, Maximilian Schiffer, Thiago Serra, and Thibaut Vidal. Optimal decision diagrams for classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 7577–7585, 2023. doi: 10.1609/aaai.v37i6.25920.

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 (CVPR) Workshops*, pp. 3498–3507, 2026.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

Jevgenij Gamper, Navid Alemi Koohbanani, Ksenija Benes, Simon Graham, Mostafa Jahanifar, Syed Ali Khurram, Ayesha Azam, Katherine Hewitt, and Nasir Rajpoot. PanNuke dataset extension, insights and baselines. *arXiv preprint arXiv:2003.10778*, 2020. doi: 10.48550/arXiv.2003.10778.

Malay K. Ganai and Andreas Kuehlmann. On-the-fly compression of logical circuits. In *International Workshop on Logic Synthesis (IWLS)*, Dana Point, CA, June 2000.

Ary L. Goldberger, Luis A. N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000. doi: 10.1161/01.CIR.101.23.e215.

Hao Hu, Marie-José Huguet, and Mohamed Siala. Optimizing binary decision diagrams with MaxSAT for classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 3767–3775, 2022. doi: 10.1609/aaai.v36i4.20291.

Mengfei Ji, Zaid Al-Ars, Peter Hofstee, Yuchun Chang, and Baolin Zhang. FPQNet: Fully pipelined and quantized CNN for ultra-low latency image classification on FPGAs using OpenCAPI. *Electronics*, 12(19):4085, 2023. doi: 10.3390/electronics12194085.

Youngsung Kim. Deep stochastic logic gate networks. *IEEE Access*, 11:122488–122501, 2023. doi: 10.1109/ACCESS.2023.3328622.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Lukas Korinek. convlogic: Community implementation of convolutional logic networks. GitHub repository: lkorinek/convlogic, 2025. GitHub repository, accessed 2026-01-26.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In *Proceedings of the 43rd Annual Design Automation Conference (DAC)*, pp. 532–535. ACM, 2006. doi: 10.1145/1146909.1147048.

George B. Moody and Roger G. Mark. The impact of the MIT-BIH arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50, 2001.

Jennifer Ngadiuba, Vladimir Loncar, Maurizio Pierini, Sioni Summers, Giuseppe Di Guglielmo, Javier Duarte, Philip Harris, Dylan Rankin, Sergo Jindariani, Mia Liu, Kevin Pedro, Nhan Tran, Edward Kreinar, Sheila Sagear, Zhenbin Wu, and Duc Hoang. Compressing deep neural networks on FPGAs to binary and ternary precision with hls4ml. *Machine Learning: Science and Technology*, 2(1):015001, 2021. doi: 10.1088/2632-2153/aba042.

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.

Felix Petersen, Hilde Kuehne, Christian Borgelt, Julian Welzel, and Stefano Ermon. Convolutional differentiable logic gate networks. In *Advances in Neural Information Processing Systems*, volume 37, pp. 121185–121203, 2024.

PIRA Project. PIRA: A research assistant. GitHub repository: AlgebraLoveme/PIRA, 2026.

Patrick Plagwitz, Frank Hannig, Jürgen Teich, and Oliver Keszöcze. To spike or not to spike? a quantitative comparison of SNN and CNN FPGA implementations, 2023.

Thorbjörn Posewsky and Daniel Ziener. Throughput optimizations for FPGA-based deep neural network inference. *Microprocessors and Microsystems*, 60:151–161, 2018. doi: 10.1016/j.micpro.2018.04.004.

Junming Qiu, Rongzhen Ye, Weilin Luo, Kunxun Qi, Hai Wan, and Yue Yu. OBDD-NET: End-to-end learning of ordered binary decision diagrams. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 2430–2439. ACM, 2025. doi: 10.1145/3746252.3761195.

Heinz Riener, Siang-Yun Lee, Alan Mishchenko, and Giovanni De Micheli. Boolean Rewriting Strikes Back: Reconvergence-Driven Windowing Meets Resynthesis. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 395–402, Taipei, Taiwan, January 2022. IEEE.

Richard L. Rudell and Alberto L. Sangiovanni-Vincentelli. Multiple-valued minimization for PLA optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5):727–750, 1987. doi: 10.1109/TCAD.1987.1270318.

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

Usman Saeed. PanNuke Segmentation U-Net. Kaggle notebook. URL `https://www.kaggle.com/code/usman29/pannuke-segmentation-unet`. Accessed September 26, 2026.

Pouya Shati, Eldan Cohen, and Sheila A. McIlraith. Sat-based learning of compact binary decision diagrams for classification. In Roland H. C. Yap (ed.), *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 33:1–33:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi: 10.4230/LIPIcs.CP.2023.33.

W. Nick Street, William H. Wolberg, and Olvi L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In *Biomedical Image Processing and Biomedical Visualization*, volume 1905 of *Proceedings of SPIE*, pp. 861–870, 1993. doi: 10.1117/12.148698.

Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, pp. 65–74, 2017. doi: 10.1145/3020078.3021744.

Yaman Umuroglu, Yash Akhauri, Nicholas J. Fraser, and Michaela Blott. High-throughput DNN inference with LogicNets. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 238. IEEE, 2020. doi: 10.1109/FCCM48280.2020.00071.

Erwei Wang, James J. Davis, Peter Y. K. Cheung, and George A. Constantinides. LUTNet: Rethinking inference in FPGA soft logic. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 26–34. IEEE, 2019.

Olivia Weng, Marta Andronic, Danial Zuberi, Jiaqing Chen, Caleb Geniesse, George A. Constantinides, Nhan Tran, Nicholas J. Fraser, Javier M. Duarte, and Ryan Kastner. Greater than the sum of its LUTs: Scaling up lut-based neural networks with AmigoLUT. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 25–35. ACM, 2025. doi: 10.1145/3706628.3708874.

William Wolberg, Olvi Mangasarian, Nick Street, and W. Street. Breast cancer wisconsin (diagnostic). UCI Machine Learning Repository, 1993. Dataset.

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. 135798–135828, 2025. doi: 10.52202/085713-4085.

Valentina Zantedeschi, Matt J. Kusner, and Vlad Niculae. Learning binary decision trees by argmin differentiation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12298–12309. PMLR, 2021.

Jinyu Zhan, Xingzhi Zhou, and Wei Jiang. Field programmable gate array-based all-layer accelerator with quantization neural networks for sustainable cyber-physical systems. *Software: Practice and Experience*, 51(11):2203–2224, 2021. doi: 10.1002/spe.2938.

A PSEUDOCODE FOR ADAPTIVE PROCEDURES

This appendix gives pseudocode for the two adaptive procedures introduced in §4. Algorithm 1 describes adaptive resampling for connection learning (§4.1), and Algorithm 2 describes adaptive discretization during training (§4.3).

Algorithm 1 Adaptive Resampling

Input: initial neuron weight $\boldsymbol{w} \in \mathbb{R}^{16}$, weight update function STEP, EMA momentum ρ, stability threshold ϵ, patience T, #triples per neuron $K = 16$.
Output: trained neuron weight $\boldsymbol{w}$
$\mu \leftarrow 0$, $c \leftarrow 0$
repeat
  $\boldsymbol{w} \leftarrow \mathrm{STEP}(\boldsymbol{w})$
  $h \leftarrow -\sum_{i=1}^{K} \tilde{\boldsymbol{w}}_i \log \tilde{\boldsymbol{w}}_i$, where $\tilde{\boldsymbol{w}}_i = \frac{\exp(\boldsymbol{w}_i)}{\sum_j \exp(\boldsymbol{w}_j)}$
  Increase c by 1 if $|\mu - h| \le \epsilon$, else set $c \leftarrow 0$
  $\mu \leftarrow \rho\mu + (1-\rho)h$
  if $c \ge T$ **then**
    dominated $\leftarrow$ check if $\exists i^*, \tilde{\boldsymbol{w}}_{i^*} \ge 0.95$
    dispersed $\leftarrow$ check if $\max_i \tilde{\boldsymbol{w}}_i \le 0.4$
    if dominated **then**
      resample all $(\boldsymbol{k}_i, \boldsymbol{p}_i, \boldsymbol{q}_i)$ except the dominant one
      reset $\boldsymbol{w}$ such that $\tilde{\boldsymbol{w}}_{i^*} = 0.9$ and $\tilde{\boldsymbol{w}}_j = \frac{0.1}{K-1}$ for $j \ne i^*$
    else if dispersed **then**
      resample all $(\boldsymbol{k}_i, \boldsymbol{p}_i, \boldsymbol{q}_i)$
      reset $\boldsymbol{w}$ such that $\tilde{\boldsymbol{w}}_i = \frac{1}{K}$ for all i
    end if
  end if
  reset $c \leftarrow 0$ if resampling is performed
until training ends

Algorithm 2 Adaptive Discretization

Input: weights $\boldsymbol{W}$ returned by Algorithm 1, weight update function STEP, EMA momentum ρ, stability threshold ϵ, patience T.
Output: discretized Boolean network
repeat
  $\boldsymbol{W} \leftarrow \mathrm{STEP}(\boldsymbol{W})$
  $l \leftarrow$ the first non-discretized layer
  Initialize $\mu_l \leftarrow 0$, $c_l \leftarrow 0$ if not initialized
  Compute average weight entropy h_l
  Increase c_l by 1 if $|\mu_l - h_l| \le \epsilon$, else set $c_l \leftarrow 0$
  $\mu_l \leftarrow \rho\mu_l + (1-\rho)h_l$
  if $c_l \ge T$ **then**
    Freeze and discretize layer l using Equation (2)
    Mark layer l as discretized
  end if
until training ends

B THERMOMETER ENCODING AND CHANNEL VISIBILITY

B.1 VISUAL EFFECT ON CIFAR-10 CHANNELS

Thermometer encoding can substantially alter the channel structure of natural images. As shown in Fig. 11, many encoded channels lose color information and preserve only partial edge patterns. Moreover, only a small subset of channels contains non-trivial, non-overlapping signal. Often, one channel carries most label-relevant information, while other channels largely repeat it. This observation motivates the channel-visibility hypothesis in §5.2.

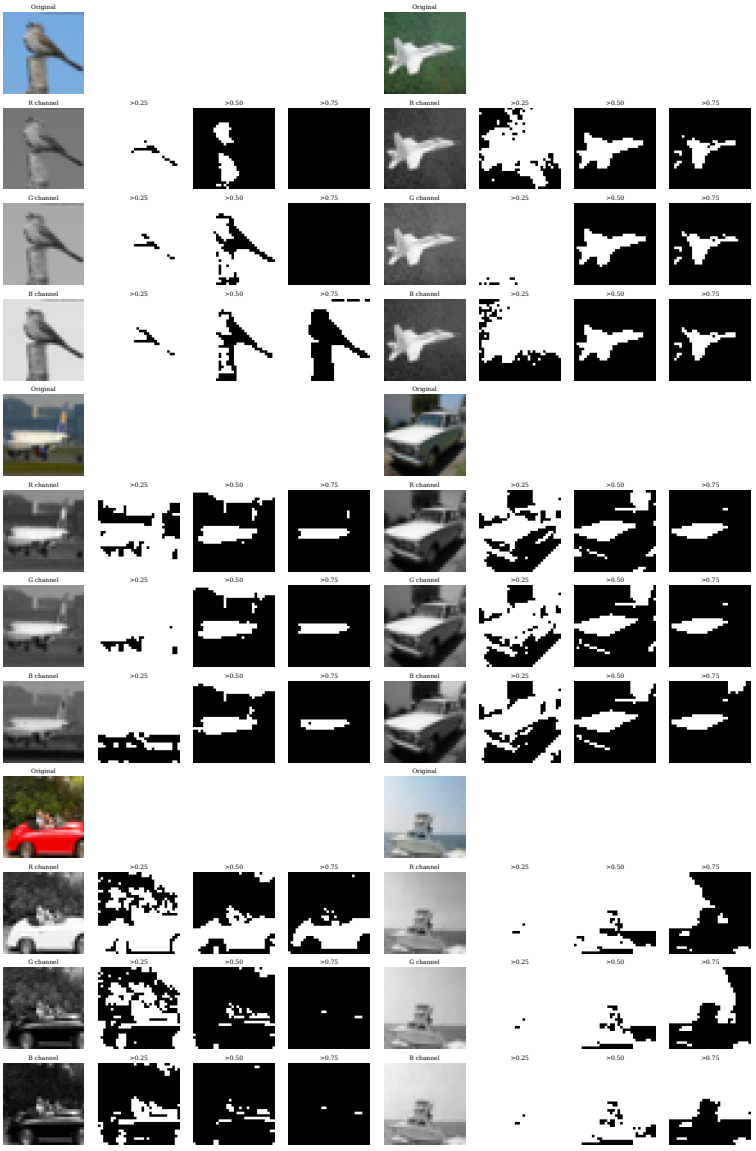


Fig. 11: Illustration of the destructive effect of thermometer encoding on the channels of natural images from CIFAR-10.

B.2 CHANNEL-VISIBILITY EXPERIMENT

To test whether the channel-visibility paradox is tied to information loss from thermometer encoding, we conduct the following three experiments.

Repeating the channel-restriction experiment on MNIST. Because thermometer encoding collapses MNIST inputs to a single Boolean channel, so it does not destroy cross-channel information. Fig. 12 shows that the paradox largely disappears: restricting channel visibility causes only negligible performance differences, rather than the substantial improvement observed on CIFAR-10. This result supports the hypothesis that the channel-visibility paradox on natural images is driven by information loss across thermometer-encoded channels.

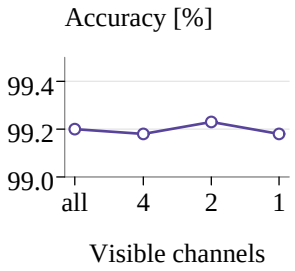


Fig. 12: Channel-visibility experiment on MNIST. Restricting channel visibility leads to only negligible performance differences, unlike the clear improvement observed on CIFAR-10.

Repeating the channel-restriction experiment on binarized CIFAR-10. We convert each CIFAR10 image to grayscale and then binarize it into black-and-white. Now the thermometer representation results in only one input channel. Our hypothesis predicts that restricting channel visibility should no longer be beneficial in this setting. The results in Table 6 are consistent with this prediction: restricting visibility to one channel is no longer optimal.

Table 6: Test accuracy on grayscale, binarized CIFAR-10 for different channel-visibility settings.

Visible channels	Acc. (%)
1	49.28
2	50.05
4	48.95
All	49.52

Sweeping through the thermometer-encoding resolutions. We test the robustness of the channel-visibility paradox to the thermometer-encoding resolution k, on the small model on CIFAR10. As shown in Table 7, restricting to one channel achieves the best accu. at all resolutions, outperforming the next-best setting by 1.21–2.49 percentage points.

Table 7: Test accuracy of the small model on CIFAR-10 for different thermometer-encoding resolutions k and channel-visibility settings.

	Acc. (%)			
k	1 channel	2 channels	4 channels	All channels
3	65.53	63.04	62.08	60.52
7	65.07	63.86	62.90	60.58
15	66.11	64.23	62.36	61.86
31	65.76	63.08	63.87	61.66

C VARIANCE ACROSS RANDOM SEEDS

To verify stability, we repeat the main connection-learning and convolution experiments from §5.1 and §5.2 across random seeds. Tables 8–11 report the corresponding means and standard deviations. Our method consistently outperforms the baselines by a clear margin, indicating that the improvements are stable across seeds.

Table 8: Statistics on MNIST for connection learning.

Method	Acc.	*#neurons*	*#BOPs*
DiffLogicNet (small)	$97.03\% \pm 0.10\%$	48.0 K	27.0 ± 0.1 K
DiffLogicNet (medium)	$98.78\% \pm 0.02\%$	384 K	215 ± 1 K
Ours (small)	$97.77\% \pm 0.13\%$	48.0 K	25.2 ± 0.4 K
Ours (medium)	$99.06\% \pm 0.03\%$	384 K	190 ± 1 K

Table 9: Statistics on CIFAR-10 for connection learning.

Method	Acc.	*#neurons*	*#BOPs*
DiffLogicNet (small)	$52.70\% \pm 0.15\%$	48.0 K	27.0 ± 0.7 K
DiffLogicNet (medium)	$59.74\% \pm 0.11\%$	512 K	311 ± 1 K
DiffLogicNet (large)	$61.48\% \pm 0.76\%$	1.28 M	780 ± 2 K
Ours (small)	$55.81\% \pm 0.33\%$	48.0 K	27.6 ± 0.3 K
Ours (medium)	$62.46\% \pm 0.27\%$	512 K	296 ± 2 K
Ours (large)	$64.16\% \pm 0.26\%$	1.28 M	655 ± 1 K

Table 10: Statistics on MNIST for convolution.

Method	Acc.	*#neurons*	*#BOPs*
TreeLogicNet-S	$97.76\% \pm 0.04\%$	185 K	120 ± 1 K
TreeLogicNet-M	$98.87\% \pm 0.12\%$	740 K	433 ± 2 K
TreeLogicNet-L	$99.28\% \pm 0.03\%$	11.8 M	6.20 ± 0.03 M
Ours-S	$99.30\% \pm 0.02\%$	260 K	131 ± 1 K
Ours-M	$99.35\% \pm 0.02\%$	520 K	249 ± 1 K

Table 11: Statistics on CIFAR-10 for convolution.

Method	Acc.	*#neurons*	*#BOPs*
TreeLogicNet-S	$59.47\% \pm 0.35\%$	874 K	538 ± 8 K
TreeLogicNet-M	$71.30\% \pm 0.06\%$	7.00 M	3.76 ± 0.06 M
Ours-T	$62.92\% \pm 0.31\%$	145 K	84.8 ± 0.7 K
Ours-S	$66.87\% \pm 0.30\%$	291 K	164 ± 1 K
Ours-M	$69.98\% \pm 0.17\%$	582 K	318 ± 1 K
Ours-L	$72.33\% \pm 0.23\%$	2.33 M	1.08 ± 0.12 M

D Additional Experiments and Ablations

D.1 MNIST Component Results

We report the MNIST counterparts of the component studies from §5.1 and §5.2 in Tables 12 and 13. The trends are consistent with the CIFAR-10 results in the main text: learned connections improve over DiffLogicNet, and the compact convolutional architecture improves over TreeLogicNet while using substantially fewer neurons.

Table 12: Comparison between DiffLogicNet and our method on MNIST.

Method	Acc.	*#neurons*	*#BOPs*
DiffLogicNet (small)	97.16%	48.0 K	26.9 K
DiffLogicNet (medium)	98.80%	384 K	214 K
Ours (small)	97.95%	48.0 K	25.0 K
Ours (medium)	99.10%	384 K	190 K

Table 13: Comparison between TreeLogicNet and our method on MNIST.

Method	Acc.	*#neurons*	*#BOPs*
TreeLogicNet-S	97.77%	185 K	117 K
TreeLogicNet-M	98.96%	740 K	427 K
TreeLogicNet-L	99.33%	11.8 M	6.20 M
Ours-S	99.15%	260 K	128 K
Ours-M	99.35%	520 K	246 K

D.2 Comparison with Fixed-Schedule Discretization Strategies

We compare the proposed adaptive discretization with two fixed-schedule discretization strategies and with a baseline that uses no training-phase discretization. All methods use the same training budget and are evaluated on two architectures.

Table 14 shows that all training-phase discretization strategies improve over the no-discretization baseline by a clear margin. A well-tuned fixed schedule can approach the performance of adaptive discretization, but the adaptive strategy performs best in all cases without requiring an additional schedule search. The difference between the "fixed every 10k" and "fixed every 20k" columns shows that fixed-schedule discretization is sensitive to this predefined schedule.

Table 14: Comparison between adaptive and fixed-schedule discretization strategies on Ours-T and Ours-S with seed variance. Reported results are test accuracy on CIFAR-10.

	Without training-phase discretization	Fixed Schedule (every 10k)	Fixed Schedule (every 20k)	Adaptive
Ours-T	60.74% $\pm$ 0.46%	62.20% $\pm$ 0.15%	61.52% $\pm$ 0.04%	62.92% $\pm$ 0.31%
Ours-S	65.27% $\pm$ 0.58%	66.57% $\pm$ 0.21%	66.24% $\pm$ 0.17%	66.87% $\pm$ 0.30%

D.3 Additional Experiments on Image Segmentation and Other Modalities

Image segmentation. We evaluate our method on histopathology image segmentation using PanNuke (Gamper et al., 2020), which provides nuclei instance-segmentation and classification annotations across multiple tissue types. We compare with a CNN-based U-Net from a public Kaggle notebook (Saeed). The U-Net uses convolutional blocks, pooling, and skip connections and achieves 82.02% pixel accuracy. Our six-layer Boolean network processes binarized images through Boolean convolutional and decoding layers without skip connections, achieving a comparable pixel accuracy of 81.91%.

Table 15 reports accuracy and inference cost. We benchmark both models on an Intel Xeon Gold 6242 CPU at 2.80 GHz with batch size 1 and 16 PyTorch intra-op threads, and on an NVIDIA GeForce RTX 2080 Ti GPU with batch size 32. Each timing result averages 100 runs after 10 untimed warm-up runs. For both models, GPU time is the amortized processing time per sample, computed by dividing the batch execution time by 32. For Boolean-network GPU inference, we use bit-packing following Petersen et al. (2022). The same Boolean activation from multiple samples is packed into distinct bits of a machine-word integer. Bitwise operations such as AND and XOR then evaluate a gate across the packed samples simultaneously, providing bit-level parallelism in addition to CUDA thread parallelism.

Table 15: Accuracy and inference cost on PanNuke image segmentation. CPU times use batch size 1. GPU times are amortized per sample over batches of 32.

Model	Accuracy	CPU Runtime (ns)	GPU Time/Sample (ns)	#BOPs	Est. FPGA Latency (ns)
U-Net	82.02%	103,155	6028	—	—
Boolean Network	81.91%	21,524	14	344K	6.32

Under these settings, the Boolean network achieves approximately $5\times$ and $430\times$ speedups over U-Net on CPU and GPU, respectively. The heuristic timing model in Appendix J estimates its FPGA latency at 6.32 ns.

Other modalities. We also evaluate on two non-image tasks: Breast Cancer classification for tabular data (Wolberg et al., 1993; Street et al., 1993) and ECG classification on the PhysioNet MIT-BIH Arrhythmia dataset for sequence data (Goldberger et al., 2000; Moody & Mark, 2001). As shown in Table 16, our method outperforms DiffLogicNet on both tasks under matched architectures, supporting the applicability of connection learning beyond vision.

Table 16: Test accuracy on the Breast Cancer and PhysioNet MIT-BIH Arrhythmia (ECG classification) datasets. We compare our method and DiffLogicNet under identical model architectures.

Dataset	Method	Acc.	*#Neurons*
Breast Cancer	DiffLogicNet	$73.33\% \pm 0.13\%$	384
	Ours	$74.28\% \pm 0.00\%$	384
ECG	DiffLogicNet	$97.53\% \pm 0.03\%$	48K
	Ours	$97.84\% \pm 0.04\%$	48K

D.4 Ablation study on the hyperparameters of the Connection-Learning Method

The two thresholds. The model is relatively insensitive to the two state thresholds within reasonable ranges, although performance deteriorates at extreme values. We thus choose the thresholds for the dominant and dispersed states as 0.95 and 0.4, respectively, and fix them globally across all experiments (i..e, without tuning them on models or datasets).

Figs. 13 and 14 report the test accuracy of the small convolutional model on CIFAR when varying one threshold at a time while keeping all other hyperparameters fixed. For the dispersed-state threshold, accuracy varies by only 0.29% over the range $[0.3, 0.5]$. For the dominant-state threshold, it varies by 0.30% over $[0.75, 0.95]$. These results indicate that model performance is stable within reasonable ranges.

Further, note that the resampling is inactive when threshold equals 0 and 1.0 for dispersed and dominant states, respectively, where the performance degrades clearly. This further supports the benefit of the proposed resampling mechanism.

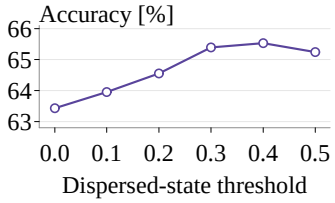


Fig. 13: Test accuracy versus the dispersed-state threshold on the small convolutional model on CIFAR-10.

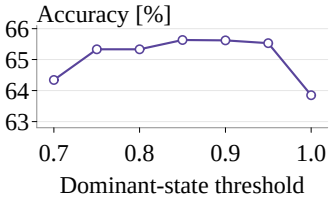


Fig. 14: Test accuracy versus the dominant-state threshold on the small convolutional model on CIFAR-10.

Patience T. Table 17 shows the model accuracy as T varies between 10 and 10^4, for the same model and dataset as above. The test performance is highest at $T = 100$ and decreases gradually as T moves away from this value. Thus, the model performance is slighter more sensitive to T than to the two state thresholds; still, the degradation remains gradual rather than abrupt.

EMA momentum. Table 18 shows the test accuracy, on the same model and dataset as above, across different values of the EMA momentum. We see the model performance is quite robust to this hyperparameter.

Table 17: Effect of resampling patience T on test accuracy of the small convolutional model on CIFAR-10.

Patience T	Acc. (%)
10	64.41
50	65.03
100	65.53
500	64.37
1000	64.23
5000	63.30
10000	62.96

Table 18: Effect of EMA momentum ρ on test accuracy of the small convolutional model on CIFAR-10.

EMA momentum ρ	Acc. (%)
0.999	65.92
0.99	65.53
0.9	65.71
0.5	65.30

E Implementation for TreeLogicNet

Since Petersen et al. (2024) did not release the official TreeLogicNet implementation, we use the community implementation from Korinek (2025). To validate this baseline, we compare its test accuracies with those reported by Petersen et al. (2024). Table 19 shows that the community implementation is slightly weaker under its default setting, but matches the reported TreeLogicNet-M

accuracy after applying the same data augmentation used in our experiments. We therefore use this implementation as the TreeLogicNet baseline for convolutional Boolean networks.

Table 19: Comparison between the accuracy reported by Petersen et al. (2024) and the community TreeLogicNet implementation, named ConvLogic.

Model	Reported by Petersen et al. (2024)	ConvLogic reported	ConvLogic + aug
TreeLogicNet-S	60.38%	59.84%	59.79%
TreeLogicNet-M	71.01%	69.15%	71.37%

F Bivariate Boolean Functions

Table 20 lists the truth table for all sixteen bivariate Boolean functions. For example, B_1 is the constant-0 function, B_2 is AND, B_4 is identity, B_8 is OR, and B_{16} is the constant-1 function.

Table 20: Truth table of the sixteen bivariate Boolean functions.

x_1	x_2	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	B_{14}	B_{15}	B_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

G Pruning Algorithm

We use the pruning algorithm in this appendix to compute the effective number of Boolean operations (#BOPs) after training. This lightweight post-processing step removes trivial and disconnected neurons from the learned network, giving a more faithful estimate of Boolean-network inference complexity before hardware synthesis.

First, we exclude constant neurons (i.e., B_1 and B_{16}), since they can share a single constant source in the circuit. We also remove identity neurons (i.e., B_4 and B_6) by rewiring their inputs, and negation neurons (i.e., B_{11} and B_{13}) by composing the negation with adjacent operations. We then remove neurons that are not connected to the final output layer, since they do not affect the network output. Algorithm 3 gives the procedure. In our experiments, we use an equivalent GPU-based implementation to compute #BOPs efficiently.

When comparing against DiffLogicNet and TreeLogicNet, we apply the same pruning procedure to all learned networks to ensure a fair comparison of #BOPs. This pruning step is intentionally simpler than full logic synthesis, which can perform more advanced hardware-level simplification and mapping. Nevertheless, the resulting #BOPs are comparable to those reported in Petersen et al. (2022; 2024), indicating that this procedure is sufficient for estimating Boolean-network complexity in the main comparisons.

Algorithm 3 Pruning

Input: trained Boolean network
Output: pruned Boolean network, #BOPs after pruning
pruned $\leftarrow$ output neurons $\cup\{0, 1\}$, BOPs $\leftarrow 0$
Q $\leftarrow$ queue(output neurons)
repeat
  current $\leftarrow$ Q.dequeue()
  if current is constant **then**
    connect to the constant neuron in the pruned network instead
  else if current is identity or negation **then**
    $n \leftarrow$ the nontrivial input neuron of current
    add n to pruned and Q, respectively, if n is not already in pruned
    if current is negation **then**
      adjust the connection from n to the neurons accepting current as their input to account for the negation
    end if
  else
    BOPs $\leftarrow$ BOPs + 1
    $n_1, n_2 \leftarrow$ two input neurons of current
    add n_1, n_2 to pruned and Q, respectively, if they are not already in pruned
  end if
until Q is empty

H Experimental Details

Setup We evaluate our method on MNIST (LeCun et al., 1998) and CIFAR-10 (Krizhevsky, 2009). For MNIST, we round floating-point pixels to the nearest Boolean values and apply random rotations up to 15° and random affine transformations with translations up to 10% of the image size in each spatial direction during training. For CIFAR-10, we use thermometer encoding with equal thresholds (see §3) to convert floating-point pixels to Boolean strings of varying lengths, following Petersen et al. (2022; 2024). For convolutional networks, these Boolean strings are concatenated along the channel dimension. During training, we apply random horizontal flips and random 32×32 crops with 2-pixel padding. No data augmentation is used at inference. Both datasets use the population-count decoder described in §3.

Baselines For connection learning and convolution design, we compare against DiffLogicNet (Petersen et al., 2022) and TreeLogicNet (Petersen et al., 2024), respectively. We use the official DiffLogicNet implementation and a community TreeLogicNet implementation that matches the reported results after applying our data augmentation protocol (see Appendix E), since the official TreeLogicNet implementation is not publicly available. For adaptive discretization, we compare against Gumbel noise injection (Yousefi et al., 2025) using the official implementation. For fairness, all methods use the same input encoding, output decoding, data augmentation, and optimizer within each comparison.

Model Architectures We list the convolutional architectures used in the experiments below. The variants T (tiny), S (small), M (medium), and L (large) correspond to $k = 64, 128, 256$, and 1024, respectively. Let N denote the number of thresholds in the thermometer encoding. For MNIST, we set $N = 1$. For CIFAR-10, we set $N = 3$ for tiny and small models, $N = 7$ for medium models, and $N = 31$ for large models.

MNIST *Model Architecture:*

- **Layer 1:** Convolutional layer with kernel size 3×3, stride 2, and padding 1, mapping $(N, 28, 28) \to (k, 14, 14)$.
- **Layer 2:** Convolutional layer with kernel size 3×3, stride 1, and padding 1, mapping $(k, 14, 14) \to (k, 14, 14)$.
- **Layer 3:** Convolutional layer with kernel size 3×3, stride 2, and padding 1, mapping $(k, 14, 14) \to (4k, 7, 7)$.

- **Layer 4:** Convolutional layer with kernel size 3×3, stride 1, and padding 1, mapping $(4k, 7, 7) \to (4k, 7, 7)$.
- **Layer 5:** Flatten operation.
- **Layer 6:** Logic layer mapping $196k \to 625k$ neurons.
- **Layer 7:** Logic layer mapping $625k \to 625k$ neurons.
- **Layer 8:** Group-sum (population count) layer with 10 output groups.

CIFAR-10 Model Architecture:

- **Layer 1:** Convolutional layer with kernel size 3×3, stride 2, and padding 1, mapping $(3N, 32, 32) \to (k, 16, 16)$.
- **Layer 2:** Convolutional layer with kernel size 3×3, stride 1, and padding 1, mapping $(k, 16, 16) \to (k, 16, 16)$.
- **Layer 3:** Convolutional layer with kernel size 3×3, stride 2, and padding 1, mapping $(k, 16, 16) \to (4k, 8, 8)$.
- **Layer 4:** Convolutional layer with kernel size 3×3, stride 1, and padding 1, mapping $(4k, 8, 8) \to (4k, 8, 8)$.
- **Layer 5:** Flatten operation.
- **Layer 6:** Logic layer mapping $256k \to 625k$ neurons.
- **Layer 7:** Logic layer mapping $625k \to 625k$ neurons.
- **Layer 8:** Group-sum (population count) layer with 10 output groups.

Training Details We initialize regular models with Gaussian weights $\mathcal{N}(0, 1)$ to match DiffLogicNet. For convolutional models, we use the residual initialization from TreeLogicNet, setting the normalized weight of B_4 to 0.9 and the normalized weights of the other Boolean functions to $0.1/15$. We also compare Gaussian and residual initialization for regular models on MNIST and CIFAR-10. As shown in Table 21, residual initialization does not consistently outperform Gaussian initialization for regular models, although it improves performance on the larger CIFAR-10 models.

Table 21: Comparison between Gaussian and residual initialization for regular models trained with our method.

Dataset	Model	Gaussian	Residual
MNIST	small	97.95%	97.89%
	medium	99.10%	98.61%
CIFAR-10	small	56.13%	56.09%
	medium	62.84%	63.52%
	large	64.53%	65.42%

All models are trained with Adam (Kingma & Ba, 2015). We use a constant learning rate of 0.01 for regular models and 0.02 for convolutional models, with batch size 128. Learning-rate scheduling did not improve performance in our experiments, potentially because improvements in the relaxed model do not necessarily translate to improvements after discretization. Regular models are trained for 300K steps and evaluated every 1K steps to select the best checkpoint. Tiny, small, and medium convolutional models are trained for 600K steps and evaluated every 2K steps. The large convolutional model is trained for 700K steps with the same evaluation frequency. We do not use weight decay.

We initialize the connection-learning parameterization as follows. At initialization, $\boldsymbol{k} = [1, \ldots, 16]$, so each neuron starts with the 16 distinct Boolean operations listed in Appendix F. For convolutional layers, we sample $(\boldsymbol{p}, \boldsymbol{q})$ uniformly with replacement from the 3×3 receptive field. For non-convolutional layers, the candidate input set is much larger, so we initialize $(\boldsymbol{p}, \boldsymbol{q})$ to encourage input coverage. Let d_{in} and d_{out} denote the input and output dimensions of the layer. We assume $2 \times d_{\text{out}} \geq d_{\text{in}}$. Otherwise, some input neurons cannot be considered by the final network. Let $r := \lfloor 2 \times d_{\text{out}} \,/\, d_{\text{in}} \rfloor$. We assign all input neurons to the first $r \times d_{\text{in}}$ input slots, with r repetitions, sample the remaining slots uniformly with replacement from all input neurons, and then shuffle the slots. This process is repeated independently 16 times to initialize each entry of $(\boldsymbol{p}, \boldsymbol{q})$. During resampling, $(\boldsymbol{p}_i, \boldsymbol{q}_i)$ are sampled uniformly from the receptive field for convolutional layers and from all input neurons for non-convolutional layers. We sample $\boldsymbol{k}_i$ uniformly from $\{1, \ldots, 16\}$.

Training times for all models are reported in Table 22.

Training Hardware All models are trained on a single RTX 2080 Ti GPU except the large convolutional model (Ours-L), which is trained on a single RTX 5090 GPU.

Hyperparameters We list detailed hyperparameter settings in Table 22. For DiffLogicNet and TreeLogicNet, we use the hyperparameters from their original implementations.

For resampling, we use an exponential moving average (EMA) with momentum $\rho = 0.99$ and stability threshold $\epsilon = 5 \times 10^{-4}$ in all experiments. The resampling patience T is tuned by model size, with larger models using larger patience values. Adaptive discretization is applied only to convolutional models, using the same EMA momentum $\rho = 0.99$ and stability threshold $\epsilon = 5 \times 10^{-4}$, with fixed patience $T = 200$.

For tiny, small, and medium convolutional models, resampling is enabled for the first $400\,\mathrm{K}$ training steps and is then disabled when adaptive discretization begins. For the large convolutional model, resampling is enabled for $500\,\mathrm{K}$ steps, followed by adaptive discretization for the remaining $200\,\mathrm{K}$ steps.

Table 22: Hyperparameter settings and training time for regular and convolutional models.

Structure	Model	Dataset	Group Sum Temp. τ	Patience T	Time (h)
Regular	Ours (small)	MNIST	10	100	2.1
	Ours (medium)	MNIST	45	1000	8.3
	Ours (small)	CIFAR-10	33	100	3.7
	Ours (medium)	CIFAR-10	100	500	10.5
	Ours (large)	CIFAR-10	100	1000	25.8
Convolutional	Ours-S	MNIST	40	100	8.2
	Ours-M	MNIST	63	100	16.0
	Ours-T	CIFAR-10	20	100	5.7
	Ours-S	CIFAR-10	40	100	9.8
	Ours-M	CIFAR-10	63	100	18.6
	Ours-L	CIFAR-10	160	15000	23.2

I GATE DISTRIBUTION

Table 23 summarizes the distributions of all 16 Boolean gates in the trained models. Two patterns suggest potential architectural priors.

Table 23: Distribution of the 16 Boolean gates in the trained convolutional models. All values are percentages.

	CIFAR-10				MNIST	
Gate	L	M	S	T	M	S
CONST0	0.1	0.1	0.0	0.1	0.5	0.3
AND	5.2	5.1	5.1	5.2	5.8	5.1
A_AND_NOT_B	5.9	5.9	6.0	6.0	7.4	7.2
A	19.5	12.2	12.3	12.0	13.6	11.7
NOT_A_AND_B	5.9	5.9	5.9	6.1	7.4	7.2
B	4.0	4.2	4.3	4.4	5.4	5.1
XOR	14.6	17.8	17.5	16.6	14.3	16.0
OR	5.5	5.8	5.9	6.0	7.3	7.6
NOR	6.0	6.3	6.0	6.2	7.6	7.7
XNOR	14.1	17.0	17.1	16.2	12.5	14.2
NOT_B	3.5	3.7	3.7	4.1	4.0	3.8
B_IMPLIES_A	4.2	4.3	4.2	4.4	3.7	3.8
NOT_A	3.5	3.7	3.8	4.0	4.0	3.9
A_IMPLIES_B	4.2	4.3	4.3	4.6	3.7	3.8
NAND	3.1	3.0	3.0	3.3	1.8	1.9
CONST1	0.1	0.0	0.0	0.0	0.1	0.0

First, XOR and XNOR together account for 26.8–34.8% of the gates across the six models. These gates can be relatively expensive in circuit implementations. For example, under an AND-inverter decomposition, an XOR or XNOR gate may require roughly 3–5 AND nodes in addition to inverters. This observation motivates adding an implementation-cost penalty for expensive gates to the continuous parameterization in Eqs. (1) and (3), consistent with the future direction of synthesis-aware training mentioned in Sec. 6.

Second, the gate distributions are approximately symmetric under swapping the two inputs, except for the identity gates A and B. For example, A_AND_NOT_B and NOT_A_AND_B occur at nearly identical frequencies in every model. The asymmetry between A and B is expected because the residual initialization assigns a dominant weight to gate A in every neuron. This pattern motivates exploring a symmetry-aware parameterization, such as retaining one representative from each input-swapped pair together with appropriate input-order handling. Such a parameterization could reduce the number of trainable parameters and therefore the training cost, although its effect on expressivity and accuracy would need to be evaluated.

J HEURISTIC FPGA TIMING ESTIMATION

We use a rough structural model to estimate FPGA latency, applying the same model and coefficients consistently to our networks and the re-evaluated TreeLogicNet baselines throughout the paper. The FPGA device used for the published baseline measurements is unavailable to us. A shared timing model provides a common basis for comparison, avoiding the confounding effects of evaluating the methods on different FPGA devices or configurations. Each network is represented as a pruned Boolean circuit followed by balanced population-count trees for class scores, with registers at the inputs and outputs. Timing is estimated after the pruning and constant simplification described in Appendix G.

Boolean and reduction depth. Let $\mathrm{pred}(v)$ denote the input nodes of a Boolean gate v. We define the depth of each circuit node recursively as

$$d(v) = \begin{cases} 0, & v \text{ is a primary input or constant,} \\ 1 + \max_{u \in \mathrm{pred}(v)} d(u), & v \text{ is a Boolean gate.} \end{cases} \tag{4}$$

For class $c \in \{1, \ldots, C\}$, let $\mathcal{S}_c$ be the multiset of nonconstant signals entering its population-count reduction. Multiplicity is retained, while constant terms contribute a fixed bias and are excluded. The Boolean depth D and balanced reduction depth L are

$$\begin{aligned} D &= \max(\{0\} \cup \{d(v) : v \in \mathcal{S}_c,\ 1 \le c \le C\}), \\ M &= \max_{1 \le c \le C} |\mathcal{S}_c|, \qquad L = \lceil \log_2 \max(1, M) \rceil . \end{aligned} \tag{5}$$

These depths capture the serial dependencies in the Boolean circuit and the balanced reduction.

Latency model. We approximate the register-to-register clock period by

$$\widehat{T} = \tau_{\mathrm{reg}} + \tau_{\mathrm{bool}} D + \tau_{\mathrm{sum}} L = (1.20 + 0.18D + 0.30L)\ \mathrm{ns}, \tag{6}$$

where τ_{reg}, τ_{bool}, and τ_{sum} represent fixed register overhead and effective delays per Boolean and reduction level. These nominal coefficients are heuristic values without device-specific calibration. The model combines depth maxima that may occur for different classes and omits routing, fan-out, placement, and adder-width effects, so its estimates depend on these simplifying assumptions and coefficient choices.

With one cycle from registered inputs to registered class scores and an initiation interval of one cycle, estimated latency is $\widehat{T}$ and estimated maximum frequency is $\widehat{f}_{\max} = 1/\widehat{T}$. Ideal throughput is one sample per cycle. Host-to-device transfers and external preprocessing are excluded.

File size and register counts. Verilog file size measures the compiled source representation rather than physical FPGA resource usage. With N_{in} input bits, C classes, and w bits per class score, the modeled flip-flop count is $N_{\mathrm{FF}} = N_{\mathrm{in}} + Cw$. This count includes the input and output registers, with no internal pipeline registers.

Additional FPGA results. Table 24 reports Boolean depth, input and output register counts, estimated maximum clock frequency, and estimated latency for all models under the same nominal timing model.

Table 24: Additional FPGA statistics on MNIST and CIFAR-10. Maximum clock frequency and latency are estimated using the same heuristic model for all models.

Dataset	Variant	Boolean depth	Total FF	$f_{\max}$ (MHz)	Latency (ns)
MNIST	TreeLogicNet-S	17	884	143.678	6.96
	TreeLogicNet-M	18	904	129.199	7.74
	TreeLogicNet-L	17	944	114.155	8.76
	Ours-S	6	914	161.812	6.18
	Ours-M	6	924	154.321	6.48
CIFAR-10	TreeLogicNet-S	22	9326	122.549	8.16
	TreeLogicNet-M	22	9356	110.375	9.06
	Ours-T	6	9336	170.068	5.88
	Ours-S	6	9346	161.812	6.18
	Ours-M	6	9356	154.321	6.48
	Ours-L	6	9376	141.243	7.08

Our Boolean depth is 6, compared with 17–18 on MNIST and 22 on CIFAR-10 for TreeLogicNet. This reduces the Boolean contribution to estimated latency, while reduction depth also affects the total.

Comparison with published FPGA timings. For the convolutional CIFAR-10 TreeLogicNet-S and TreeLogicNet-M models, our simulated runtimes are 8.16 ns and 9.06 ns, respectively, very similar to the 9 ns reported by Petersen et al. (2024) for both models. The numerical agreement provides a consistency check on the accuracy of our timing simulation for these configurations.