# LILogic Net: Compact Logic Gate Networks with Learnable Connectivity for Efficient Hardware Deployment

Katarzyna Fojcik[1] Renaldas Zioma[2] Jogundas Armaitis[2]
katarzyna.fojcik@pwr.edu.pl rz@TranscendentLogic.com ja@TranscendentLogic.com
[1]Wroclaw University of Science and Technology [2]Transcendent Logic

## Abstract

*Efficient deployment of machine learning models ultimately requires taking hardware constraints into account. The binary logic gate is the fundamental building block of all digital chips. Designing models that operate directly on these units enables energy-efficient computation. Recent work has demonstrated the feasibility of training randomly connected networks of binary logic gates (such as OR and NAND) using gradient-based methods. We extend this approach by using gradient descent not only to select the logic gates but also to optimize their interconnections (the connectome). Optimizing the connections allows us to substantially reduce the number of logic gates required to fit a particular dataset. Our implementation is efficient both at training and inference: for instance, our LILogicNet model with only 8,000 gates can be trained on MNIST in under 5 minutes and achieves 98.45% test accuracy, matching the performance of state-of-the-art models that require at least two orders of magnitude more gates. Moreover, for our largest architecture with 256,000 gates, LILogicNet achieves 60.98% test accuracy on CIFAR-10 exceeding the performance of prior logic-gate-based models with a comparable gate budget. At inference time, the fully binarized model operates with minimal compute overhead, making it exceptionally efficient and well suited for deployment on low-power digital hardware.*

## 1. Introduction

In recent years, scaling up machine learning models has driven major advances across real-world applications, enabled by specialized datacenter hardware [11]. In such environments, hardware cost and energy consumption are typically secondary concerns. As machine learning matures, however, there is growing interest in deploying models on low-power, cost-sensitive edge devices [8], demanding tighter co-design between model architecture and hardware capabilities to extend AI beyond traditional datacenter settings [2].

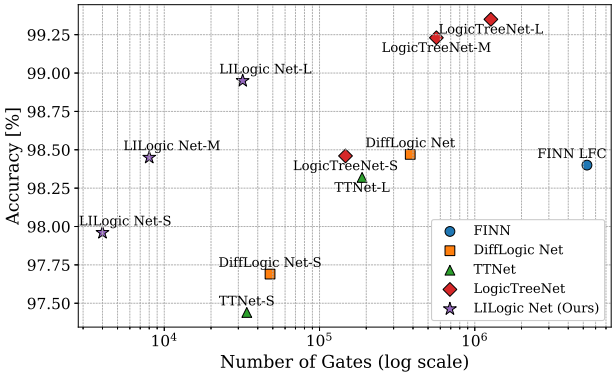



Figure 1. Gate count vs. accuracy on the MNIST dataset. Our models LILogic Nets achieve significantly higher accuracy for a given gate budget compared to state-of-the-art baselines, lying well above the Pareto front.

Up to now, machine learning models have largely relied on matrix multiplication operations, which are efficiently supported by high-level APIs and parallel processing on hardware such as GPUs [6, 30]. In contrast, most computational hardware fundamentally operates on binary logical gate operations. These binary operations are accessible in a straightforward manner on virtually all hardware substrates, including ASICs, FPGAs [15, 21], CPUs, and even microcontrollers [4, 19, 28]. To exploit this dominance of binary computation, it is desirable to design models that natively operate on binary signals and logical gates, while maintaining performance competitive with matrix-based counterparts.

One way to achieve such an evolution may be to engineer advantageous replacements for components of state-of-the-art machine learning models, such as dense fully connected neuron layers (multilayer perceptrons or MLPs)[7] and convolution layers [16, 40]. It has already been shown that Logical Gate Networks (LGNs), consisting of randomly-wired layers of trainable two-input single-output logical gates, can be trained end-to-end using gradient descent [22] and approach state-of-the-art results on various tasks, including computer vision and tabular. These promising initial results

relied on frozen connectomes, while the reported models, e.g., used $10^4$–$10^5$ gates for fitting MNIST [17, 22] and $10^4$–$10^6$ for fitting CIFAR-10 [14, 22]. However, if MLP parts of today's deep networks are to be replaced by LGNs, it is paramount to better understand and further optimize these LGNs. To that end, we treat the connectome itself as a differentiable, optimizable object.

Training of LGN connectomes has also been explored, though only for tabular datasets and primarily from an interpretability perspective [37, 38]. Yue and Jha conducted extensive ablation studies covering input preprocessing, straight-through estimators, Gumbel-softmax relaxation [20], and other training components. However, their work was limited to relatively small LGNs—up to 2,000 gates—citing training slowdowns due to computational burden, though without providing numerical measurements of training time [38]. In contrast, our focus is on scaling up LGNs and ensuring performant training at larger sizes, up to two orders of magnitude greater in gate count. To this end, we measure training efficiency and provide an open-source implementation of our full framework.

We summarize our contributions as follows:

- We propose an open framework for jointly training both the connectome and logical gate operations of a Logical Gate Network (LGN) via gradient descent.
- Our framework includes a performant, cross-platform implementation of the logical gate optimization module.
- As a case study, we introduce **LILogic Net** (*Learnable Interconnect Logic Network*), a highly compact LGN that achieves competitive accuracy on both MNIST and CIFAR-10 while requiring minimal computational cost.

Our results demonstrate that LGNs can successfully learn complex tasks when their connectomes are made learnable. This suggests the potential for LGNs to serve as hardware-native alternatives to traditional matrix-multiplication-based models. Our LILogic Net achieves an average accuracy of 98.45% on MNIST using only 8,000 logic gates—at least two orders of magnitude fewer operations than comparable state-of-the-art models (see Figure 1). On CIFAR-10, our model does not reach state-of-the-art accuracy but achieves 60.98% for the largest gate budget, outperforming other LGNs that use roughly an order of magnitude more gates, highlighting the effectiveness of learnable connectivity.

## 2. Related Work

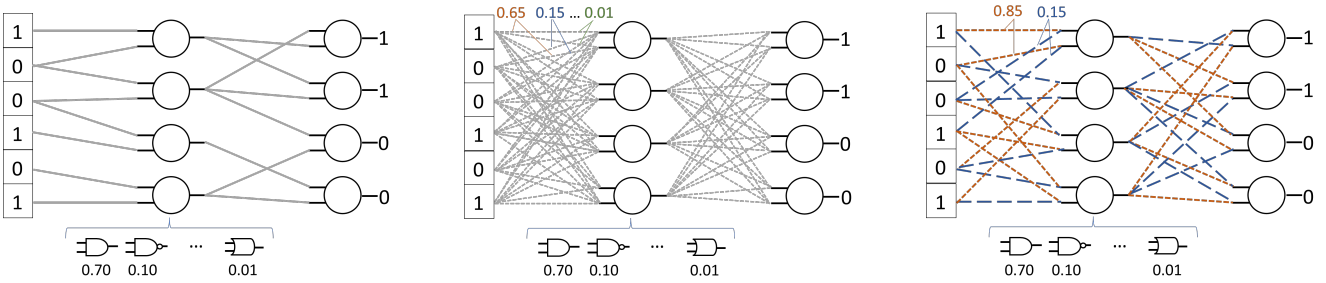
Several key trends have emerged in related areas, particularly in sparsity-aware, binarized, and hardware-efficient neural models. Binary Neural Networks (BNNs) [36], such as XNOR-Net [25] and BinaryNet [9], showed that binary weights and activations can significantly reduce memory and computation costs with minimal accuracy loss. In parallel, Quantized Neural Networks [10] reinforced the promise of low-precision optimization. IR-Net [24] addressed information loss in binary training via forward-backward techniques like Libra Parameter Binarization and Error Decay Estimators. ReActNet [18] improved binary model accuracy through generalized activations. From a deployment perspective, FINN [31] introduced a scalable FPGA framework for real-time BNN inference.

A related but conceptually distinct direction stems from differentiable logic-based neural architectures, which structure computation around learnable logic gates instead of weighted connections. Logic Gate Networks (LGNs), introduced by Petersen et al. [22], demonstrated that logic gate arrays—traditionally considered incompatible with gradient descent—can be relaxed and trained using standard deep learning methods. Even in their early form, LGNs offered benefits over BNNs in terms of architectural simplicity, interpretability, and hardware alignment. Subsequent work explored both theoretical and practical aspects of LGNs. Yue and Jha [37, 38] investigated interpretability in tabular data, employing straight-through estimators, Gumbel-softmax, and various preprocessing schemes. However, these early LGNs were limited in size and not performance-optimized. Recent work has addressed these limitations. Yousefi et al. [35] improved training stability and gate utilization by mitigating the discretization gap using Gumbel noise and Hessian regularization. Petersen et al. [23] proposed TreeLogicNet, incorporating logic gates into convolutional windows with logical $OR$ pooling, achieving 99.35% accuracy on MNIST and 86.29% accuracy on CIFAR-10—but at the cost of using 2–4 orders of magnitude more gates than our method. Although based on logical operations, TreeLogicNet retains CNN-like receptive fields and parameter sharing, resulting in higher gate and operation counts. In contrast, our design uses only logical gates, with gate count directly reflecting computational effort, enabling greater efficiency. Other contributions include the eXpLogic framework by Wormald et al.[34], which enhances interpretability through saliency maps and enables gate pruning with minimal accuracy drop. Kresse et al.[13] showed that LGNs are inherently suitable for formal verification using SAT solvers, allowing proofs of fairness and robustness. Clester [5] explored LGNs for generative applications, such as music synthesis. On the hardware side, Wang et al. [33] introduced four custom RISC-V instructions to accelerate LGN inference. Their design achieved 95.8% accuracy on MNIST using a compact network of 18,000 logic gates and reduced runtime by over 87% compared to a standard RV32IM core.

## 3. Method

### 3.1. Differentiable Logical Gate Networks

Traditional Logic Gate Networks (LGNs) are sparse, weightless models built from binary logic gates (e.g., AND, XOR). While highly efficient for inference, their discrete nature




(a) Fixed Connectome (b) Fully Learnable Dense Connectome (c) Top-K Sparse Connectome (here K=2)

Figure 2. Overview of training-time interconnect strategies.

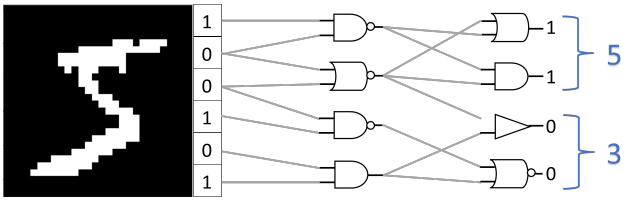



Figure 3. Inference pipeline: all logic gates and connections are fully binarized, yielding a compact, deterministic circuit.

prevents gradient-based training, thus limiting scalability. The Differentiable LGN framework introduced by Petersen et al. [22] solves this by: (i) relaxing binary inputs to continuous values in $[0, 1]$, (ii) using probabilistic interpretations of logic gates (e.g., $A \wedge B \approx A \cdot B$ [32, 39]), and (iii) representing each node as a soft combination over all 16 2-input Boolean functions, enabling learning via gradient descent.

This makes LGNs trainable with standard optimizers (e.g., Adam [12]), and—after training—easily discretizable to efficient, purely combinatorial LGNs, which means that they have no "memory" or "state".

We design an LGN architecture that jointly learns the logic gate behavior and the structure of connections between them (the connectome), enabling both efficient training and inference. Our key methodological contributions are twofold: (i) a more GPU-efficient gate representation, and (ii) a systematic comparison of several strategies for interconnect optimization.

### 3.2. Model Architecture

Our models have 1–4 logic gate layers, each containing a fixed number of gates selected from $\{2\text{K}, 4\text{K}, 8\text{K}, 16\text{K}, 32\text{K}, 64\text{K}, 128\text{K}\}$. Each node receives two relaxed binary inputs and implements one of the 16 Boolean functions [26]. All layers share the same width and interconnect strategy. We consider three strategies for wiring between layers (Figure 2):

- **(F) Fixed connectome:** Each gate receives inputs from a random subset of the previous layer. This mirrors the original Differentiable LGN [22] and serves as a baseline, typically requiring deeper or wider layers to match accuracy.
- **(L) Fully learnable dense connectome:** For each node, both of its two inputs are connected to all outputs of the previous layer (or to the input vector for the first layer). The connection weights for each input are parameterized separately via a differentiable softmax.
- **(Top-K) Sparse connectome:** For each node, both of its two inputs independently select $K$ candidate connections from the previous layer (or the input vector for the first layer). During training, a softmax over the $K$ candidates produces a weighted combination for each input. We experiment with $K \in \{2, 4, 8, 16, 32, 64, 128\}$ to trade off sparsity and expressiveness.

After training, all models are fully binarized. Each gate selects its most probable logic gate and inputs (for $L$ and Top-$K$ connectomes), yielding a fixed-size, deterministic binary circuit suitable for efficient inference (Figure 3).

### 3.3. Dataset and Preprocessing

We evaluated our models on the MNIST and CIFAR-10 datasets. The MNIST dataset consists of 60,000 training and 10,000 test images of handwritten digits [17]. Each grayscale image of size $28 \times 28$ was flattened into a 784-dimensional input vector and binarized by applying a fixed intensity threshold of 0.25. The CIFAR-10 dataset contains 50,000 training and 10,000 test color images of $32 \times 32$ pixels across ten object categories [14]. For CIFAR-10, each RGB channel was processed independently and binarized using fixed thresholds $[0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875]$. The resulting binary maps from all channels were concatenated into a single flattened vector, yielding an effective input dimensionality of 21,504.

For data augmentation, we applied dataset-specific transformations to improve generalization [29]. For MNIST, we used the `torchvision.transforms.v2` interface to apply random rotation ($\pm 10°$), shear (up to $10°$), scaling (by $\pm 10\%$), and elastic deformations with $\alpha = 64$ and $\sigma = 6$. Each augmented transformation was applied independently

to generate 10 distinct copies of the training dataset, leading to a $10\times$ expanded training set. For CIFAR-10, we applied standard natural image augmentations [27], including random cropping to $32 \times 32$ with 4-pixel reflection padding and random horizontal flips with 50% probability, expanding the training set $8\times$. We note that the original DiffLogic Net baseline [22] does not use augmentations. To enable a fair comparison and isolate the effect of learnable connectivity, we additionally perform evaluations of our models against versions with fixed connectomes under the same augmentation settings.

The validation set was constructed by applying only resizing and binarization (i.e., no augmentation) to the original training images, and the same preprocessing was used for the test set. A fixed random seed ensured reproducibility of the dataset splits and augmentation pipeline.

### 3.4. Training and Inference

All models were trained for 200 epochs using the Adam optimizer [12] with a learning rate of 0.075, a batch size of 256, and no additional regularization such as dropout or weight decay. Our experiments focus purely on the architectural and interconnect design choices, especially different sparsity regimes, and aim to isolate their effect without the influence of external regularizers. Top-$K$ interconnect variants act implicitly as a form of regularization by constraining the connectivity space.

#### 3.4.1. Timing measurements

Reported training times include the full training procedure with validation but exclude the initial data loading and the online generation of augmented samples. This ensures that the reported times reflect only the model's forward and backward passes during training.

#### 3.4.2. Differentiable Relaxation

To enable gradient-based training of logic gate selection, each node is parameterized by a vector of 16 real-valued weights, $\mathbf{w} \in \mathbb{R}^{16}$, representing logits over the space of all 16 possible binary logic gates. These logits are transformed into a probability distribution via the softmax function, scaled by a temperature parameter $\tau_g$ [1], which we kept fixed at $\tau_g = 1$ in all our experiments:

$$\mathbf{p} = \text{softmax}(\tau_g \cdot \mathbf{w}), \quad p_i = \frac{\exp(\tau_g \cdot w_i)}{\sum_j \exp(\tau_g \cdot w_j)}. \quad (1)$$

The resulting probability vector $\mathbf{p} \in \Delta^{15}$ lies in the 15-dimensional probability simplex, meaning that all entries are non-negative and sum to 1: $\sum_{i=1}^{16} p_i = 1, p_i \geq 0$. This formulation treats gate selection as a categorical distribution that can be trained end-to-end using standard backpropagation. Instead of evaluating each of the 16 gates individually (e.g., $A \cdot B$ for $A \wedge B$), we project this softmax-normalized gate distribution into a lower-dimensional functional basis space, enabling more efficient computation and improved interpretability. Specifically, the vector $\mathbf{p}$ is linearly projected via a fixed matrix $W_{16\to4} \in \mathbb{R}^{4\times16}$, yielding:

$$\mathbf{c} = W_{16\to4} \cdot \mathbf{p}, \quad \text{where} \quad \mathbf{c} = [c_1, c_2, c_3, c_4]^\top \quad (2)$$

are the coefficients for the basis $\{1, A, B, A \cdot B\}$.

The gate's continuous output is then computed as:

$$\text{output}_g = c_1 + c_2 A + c_3 B + c_4 (A \cdot B). \quad (3)$$

The fixed projection matrix $W_{16\to4}$ is given by:

$$W_{16\to4}^\top = \left[\begin{array}{r|cccc} \text{Operator} & 1 & A & B & A \cdot B \\ \hline \text{False} & 0 & 0 & 0 & 0 \\ A \wedge B & 0 & 0 & 0 & 1 \\ \neg(A \Rightarrow B) & 0 & 1 & 0 & -1 \\ A & 0 & 1 & 0 & 0 \\ \neg(A \Leftarrow B) & 0 & 0 & 1 & -1 \\ B & 0 & 0 & 1 & 0 \\ A \oplus B & 0 & 1 & 1 & -2 \\ A \vee B & 0 & 1 & 1 & -1 \\ \neg(A \vee B) & 1 & -1 & -1 & 1 \\ \neg(A \oplus B) & 1 & -1 & -1 & 2 \\ \neg B & 1 & 0 & -1 & 0 \\ A \Leftarrow B & 1 & 0 & -1 & 1 \\ \neg A & 1 & -1 & 0 & 0 \\ A \Rightarrow B & 1 & -1 & 0 & 1 \\ \neg(A \wedge B) & 1 & 0 & 0 & -1 \\ \text{True} & 1 & 0 & 0 & 0 \end{array}\right] \quad (4)$$

This formulation offers several advantages. First, it allows efficient parallel computation using dense matrix operations, leveraging GPU acceleration. Implementation does not require platform specific (CUDA) code thus it is more portable. Second, it avoids evaluating 16 separate Boolean functions for each gate, improving training speed and reducing memory usage. Third, the decomposition into basis functions enhances interpretability by exposing the contribution of each term (e.g., $A$ vs. $A \cdot B$). Finally, it provides a flexible foundation for extending the logical expressivity to higher-order gates in future work.

#### 3.4.3. Learnable Interconnections

For the learnable interconnect strategies (Top-$K$ and $L$), we similarly employ a softmax distribution over input weights to determine input pairings for each gate. For Top-$K$, a fixed number of candidates $K$ are preselected per gate at initialization, and only the softmax weights over those connections are trained. For the Fully Learnable strategy, the softmax is applied across all possible inputs, resulting in higher memory and computation costs.

To control the softness of this selection process, we introduce a separate temperature parameter $\tau_c$ for connections.

However, in our experiments we kept $\tau_c = 1$ and varied sparsity directly using the K value, which provides more explicit control.

#### 3.4.4. Binarization

After training, we convert the network into a discrete, inference-efficient form by binarizing both gate selection and interconnections. This is done by selecting the maximum probability option (the mode) from the softmax distributions. In practice, this means that each logic gate uses a fixed pair of binary inputs and computes a single Boolean function. Regardless of the training strategy, every node uses exactly two binary inputs at inference time, allowing the model to operate entirely in Boolean logic without any floating-point computation. This discretization ensures that the model becomes extremely fast and memory-efficient during inference, benefiting applications in embedded or hardware-constrained environments.

#### 3.4.5. Classification Strategy

We adopt a probabilistic majority-voting scheme for classification. The output layer consists of $n$ binary logic gates (e.g., 1,000), evenly divided into $k$ groups corresponding to the $k$ output classes (e.g., 10). During inference, each group counts the number of '1' activations, and the predicted class is the one with the highest count. During training, soft outputs are aggregated per group, and the model is optimized using softmax cross-entropy loss. The sharpness of the soft logic activations is controlled by a global temperature parameter $\tau$, which was empirically tuned for each layer width. The parameter $\tau$ affects both convergence and final accuracy, and was selected based on the results of a parameter sweep (see Sec. 4.2.4). We use classification accuracy as the evaluation metric, as MNIST and CIFAR-10 are balanced and accuracy effectively captures model performance.

## 4. Experiments

### 4.1. Training Time Comparison: Full Gate Evaluation vs. Basis Projection

To assess the computational benefits of the proposed projection-based gate evaluation, we compare the **total training time** on the MNIST dataset (for an NVIDIA A4000 GPU) between two strategies:

- **Full Gate Evaluation (FullEval):** Each of the 16 logic gates is evaluated individually per node.
- **Basis Projection (BasisProj):** A single projection into the $\{1, A, B, A \cdot B\}$ basis using a fixed matrix $W_{16\to4}$ replaces per-gate evaluation.

We evaluate six configurations, defined by the number of fixed connectome layers (1F, 2F, 3F) and layer width (8,000 and 32,000 nodes).

As shown in Table 1, using basis projection significantly reduces total training time across all tested configurations.

| Width | Type | FullEval | BasisProj | Speed-Up |
|---|---|---|---|---|
| 8,000 | 1F | 2.13 | 1.13 | 1.89× |
| | 2F | 4.75 | 1.60 | 2.96× |
| | 3F | 7.47 | 2.21 | 3.38× |
| 32,000 | 1F | 9.97 | 2.92 | 3.41× |
| | 2F | 20.78 | 5.44 | 3.82× |
| | 3F | 31.63 | 7.99 | 3.96× |

Table 1. Full training time (in minutes) and speed-up using basis projection over full gate evaluation. Results are shown for the MNIST and averaged over 10 runs with $\leq 0.1$ min variability.

The benefit becomes increasingly pronounced with deeper and wider architectures. For instance, in the 32,000 / 3F configuration, the projection-based approach completes training nearly **4× faster** than the full gate evaluation. This speed-up stems from avoiding 16 separate logic gate computations per node in favor of a single dense matrix-vector multiplication—an operation highly optimized for modern accelerators. Overall, these results demonstrate that our differentiable projection-based formulation is not only interpretable and expressive but also scalable and efficient for large models.

### 4.2. Interconnect Strategy Analysis

We performed extensive experiments on the MNIST dataset using architectures with 1–4 layers and widths ranging from 2K to 32K, exploring various connectome variants. MNIST's low computational cost allowed us to thoroughly analyze these designs, providing insights that informed the selection of a reduced set of architectures for CIFAR-10 experiments. Test accuracy was measured at binarized inference, averaged over 10 runs for 1–2 layer models and 5 runs for 3–4 layer models, with per-run statistics and training times reported in the appendix. All MNIST experiments were conducted on an NVIDIA A4000 GPU.

#### 4.2.1. Empirical Comparison Across Layer Widths and Depths on MNIST

As shown in Table 2, random and non-trainable $F$ interconnects, consistently perform the worst. Their inflexibility requires significantly wider or deeper networks to achieve moderate accuracy, illustrating the limitations of static wiring. Top-$K$ sparse interconnects, on the other hand, deliver strong results across nearly all configurations. Even under extreme sparsity (e.g., Top8), models reach over **98.6%** accuracy when wide or deep enough, outperforming $L$ interconnects in many regimes. The best-performing configuration overall—a Top32 model with two 16K-wide layers (2Top32-16K)—achieves **98.95%** accuracy. Notably, Top-$K$ models continue to improve with additional depth, unlike $L$-type architectures, whose performance tends to saturate or decline beyond two layers due to overfitting or training instability.

| Layer Width | Layer Type | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 2,000 | F | 87.09 | 90.82 | 93.69 | 95.38 |
| | Top8 | 94.29 | 96.97 | 97.47 | 97.59 |
| | Top32 | 96.38 | 97.50 | **97.83** | **97.71** |
| | Top128 | 97.02 | **97.66** | 97.67 | 96.92 |
| | L | **97.25** | 97.37 | 96.13 | 92.00 |
| 4,000 | F | 90.16 | 93.28 | 95.77 | 96.90 |
| | Top8 | 96.27 | 97.91 | 98.13 | 98.22 |
| | Top32 | 97.54 | 98.17 | **98.38** | **98.29** |
| | Top128 | 97.94 | **98.28** | 98.30 | 97.56 |
| | L | **97.96** | 98.11 | 97.02 | 94.41 |
| 8,000 | F | 91.48 | 94.83 | 96.83 | 97.69 |
| | Top8 | 97.17 | 98.43 | 98.60 | 98.63 |
| | Top32 | 98.15 | 98.63 | **98.77** | **98.72** |
| | Top128 | 98.39 | **98.67** | 98.72 | 97.94 |
| | L | **98.45** | 98.62 | 97.87 | 96.81 |
| 16,000 | F | 93.16 | 96.43 | 97.82 | 98.34 |
| | Top8 | 98.15 | 98.74 | 98.80 | **98.87** |
| | Top32 | 98.50 | **98.95** | **98.86** | 98.73 |
| | Top128 | 98.55 | 98.89 | 98.84 | 98.23 |
| | L | **98.58** | 98.82 | - | - |
| 32,000 | F | 94.74 | 97.41 | 98.27 | 98.52 |
| | Top8 | 98.41 | 98.75 | 98.78 | 98.76 |
| | Top32 | **98.58** | 98.83 | **98.84** | **98.89** |
| | Top128 | 98.51 | **98.91** | 98.83 | 97.77 |
| | L | 98.52 | - | - | - |

Table 2. Test accuracy results on MNIST across different layer widths and number of layers for various gating methods. The results are averaged over 10 runs for 1-2 layer models and 5 runs for 3-4 layer models, all with $\leq 0.1\%$ variation. Bolded values indicate the best performance per block.

Figure 4 illustrates how test accuracy scales with depth for different interconnect strategies and layer widths. Notably, even under aggressive sparsity (Top8), models consistently outperform $F$ interconnects across all depths—even with reduced layer width. For instance, a 2Top8-2K model achieves 96.97%, surpassing the 2F-8K model (94.83%). $L$ interconnects attain slightly higher accuracy at shallow depths (e.g., 98.45% for 1L-8K), but their gains saturate quickly. In contrast, Top8 models continue to improve with added depth, highlighting that sparse, learnable connectivity not only generalizes well but also scales more effectively than static or dense alternatives.

#### 4.2.2. Convergence of Top-K to Fully Learnable Connectivity

Figure 5 presents a representative case on the MNIST dataset of a 1-layer model with 2K gates, illustrating how Top-$K$ sparse interconnect performance improves with $K$, approaching that of the fully learnable (L) connectome. Accuracy rises quickly for small $K$ and saturates as $K$ increases; a logistic fit in $\log_2(K)$ space captures this trend. For $K \geq 128$, performance nearly reaches the fully learnable ceiling, reflecting the general trade-off between sparsity and accuracy.

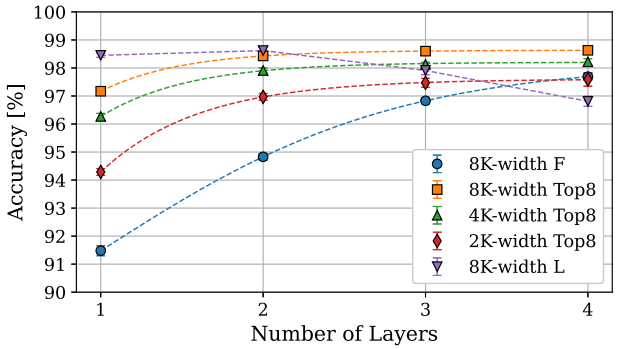



Figure 4. Test accuracy vs. depth for various interconnects.

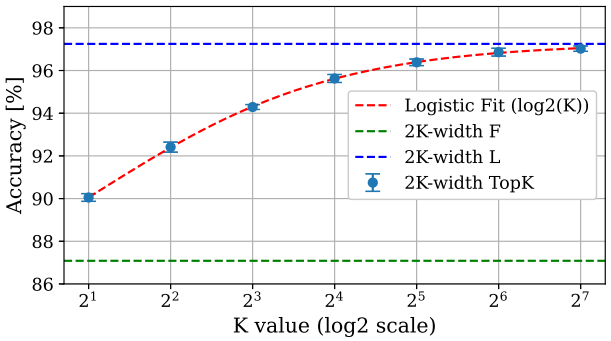



Figure 5. Convergence of 2K-width Top-$K$ test accuracy to $L$-type connectivity as $K$ increases.

#### 4.2.3. Sparse Interconnects as Compositional Depth

Sparse interconnect strategies such as Top-$K$ restrict each logic gate to operate on only a small subset of input pairs (e.g., 8 for Top8). While this limits the expressivity of a single layer, stacking layers expands the effective receptive field. In principle, an $N$-layer Top-$K$ network can access up to $K^N$ input paths—an upper bound that reflects the combinatorial breadth introduced by depth, even though each gate receives only two inputs at inference. Thus, depth enables sparse LGNs to recover some of the flexibility of denser interconnects at much lower cost. For example, two Top8 layers can, in the best case, reach up to 64 input paths, though the learned wiring may cover only part of this space. Depth therefore serves as an effective way to capture richer dependencies while maintaining hardware-efficient sparse connectivity.

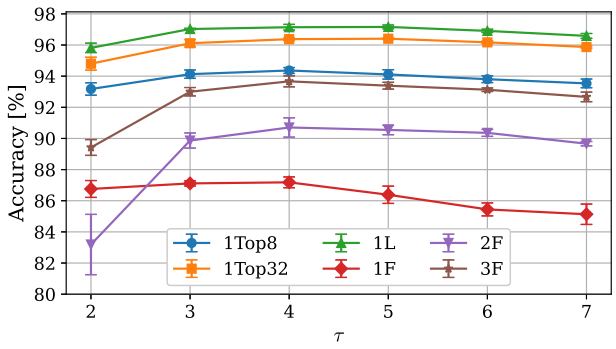



Figure 6. Test accuracy as a function of $\tau$ for 2K-width models. Used as a reference for choosing temperature values.

#### 4.2.4. Sensitivity to Temperature Parameters

We explored the impact of the global softmax temperature $\tau$ and observed that it needs to scale with layer width. Based on trends exemplified in Figure 6 (for 2K-width architectures on MNIST), we selected $\tau = 4$ for 2K layers, $\tau = 5$ for 4K layers, and $\tau = 10$ for 8K, 16K, and 32K layers. For the more complex CIFAR-10 dataset, higher values were required, with $\tau = 20$ for 8K layers, $\tau = 90$ for 64K and $\tau = 100$ for 128K layers. These values were empirically chosen to balance smooth gradient flow during training and confident gate assignments post-binarization. Lower values caused overly sharp distributions too early, leading to vanishing gradients, while excessively high values resulted in unstable training due to overly flat distributions.

### 4.3. Trade-offs Between Depth and Top-K Sparsity Under Gate Budget Constraints

To assess architectural efficiency under realistic hardware constraints, we evaluate models with equal total gate counts—a key factor when logic-level resources and inference-time connectivity are fixed. Table 3 presents a comparison of the top-performing Top-$K$ against $L$ connectome (selected from Table 2) , across gate budgets ranging from 2K to 32K. We analyze the impact of depth, width, and interconnect strategy on test accuracy and training time.

A similar comparison for the CIFAR-10 dataset is included in the appendix, focusing on a more selective set of architectures informed by insights derived from the comprehensive MNIST experiments.

#### 4.3.1. Effect of Layer Configuration at Fixed Budgets on MNIST

At small budgets (2K–8K gates), $L$ architectures perform surprisingly well. For example, the 1L–4K model achieves 97.96% accuracy, outperforming deeper or sparser Top-$K$ variants of the same size (e.g., 2Top128–2K with 97.66%). These results suggest that at limited scale, increasing width offers more gain than depth, and dense optimization helps make better use of constrained capacity. However, as gate budgets grow, Top-$K$ models begin to dominate. For instance, the 2Top128–8K configuration reaches 98.67%, outperforming both 2L–8K (98.62%) and 1L–16K (98.58%).

| #gates | Model | Layer Width | Test Acc.[%] | Train Time [min] |
|---|---|---|---|---|
| 2,000 | 1Top128 | 2,000 | 97.02 | 5.6 |
| | 1L | 2,000 | **97.25** | 5.0 |
| 4,000 | 2Top128 | 2,000 | 97.66 | 17.6 |
| | 2L | 2,000 | 97.37 | 4.1 |
| | 1Top128 | 4,000 | 97.94 | 8.0 |
| | 1L | 4,000 | **97.96** | 2.5 |
| 8,000 | 4Top32 | 2,000 | 97.71 | 10.8 |
| | 4L | 2,000 | 92.00 | 8.5 |
| | 2Top128 | 4,000 | 98.28 | 33.1 |
| | 2L | 4,000 | 98.11 | 10.8 |
| | 1Top128 | 8,000 | 98.39 | 16.2 |
| | 1L | 8,000 | **98.45** | 4.3 |
| 16,000 | 4Top32 | 4,000 | 98.29 | 19.4 |
| | 4L | 4,000 | 94.41 | 27.4 |
| | 2Top128 | 8,000 | **98.67** | 82.4 |
| | 2L | 8,000 | 98.62 | 4.4 |
| | 1Top128 | 16,000 | 98.55 | 42.3 |
| | 1L | 16,000 | 98.58 | 7.8 |
| 32,000 | 4Top32 | 8,000 | 98.72 | 55.5 |
| | 4L | 8,000 | 96.81 | 101.4 |
| | 2Top32 | 16,000 | **98.95** | 57.1 |
| | 2L | 16,000 | 98.82 | 154.9 |
| | 1Top32 | 32,000 | 98.58 | 24.2 |
| | 1L | 32,000 | 98.52 | 15.0 |

Table 3. Test accuracy for different total gates counts. Bold indicates best in each block.

A key observation is that 1-layer and 2-layer models frequently outperform 4-layer ones of the same gate count (e.g., 2Top128–8K vs. 4Top32–4K), indicating diminishing returns from excessive depth when width and sparsity are appropriately balanced. However, 1-layer wide models often underperform their 2-layer equivalents at higher budgets, indicating that once sufficient gates are available, added depth becomes beneficial.

Interestingly, denser L-type models often train faster than sparse Top-$K$ counterparts (e.g., 1L–8K takes 4.3 min vs. 1Top128–8K at 16.2 min). Although fully learnable interconnects are theoretically more memory- and compute-intensive, this runtime difference likely arises from more efficient GPU support for dense matrix operations compared to sparse, index-heavy patterns.

| Method | Acc [%] | #Gates | ΔAcc/G [$\times 10^{-5}$] |
|---|---|---|---|
| FINN LFC [31] | 98.40 | 5.28 M | 0.16 |
| DiffLogic Net-S [22] | 97.69 | 48 K | 16.02 |
| DiffLogic Net [22] | 98.47 | 384 K | 2.21 |
| LogicTreeNet-S [23] | 98.46 | 147 K | 5.76 |
| LogicTreeNet-M [23] | 99.23 | 566 K | 1.63 |
| LogicTreeNet-L [23] | 99.35 | 1.27 M | 0.74 |
| TTNet-S [3] | 97.44 | 34 K | 21.88 |
| TTNet-L [3] | 98.32 | 188 K | 4.43 |
| RV-LGN [33] | 95.80 | 18 K | 32.22 |
| eXpLogic [34] | 91.20 | 5 K | 24.00 |
| **LILogicNet-S (ours)** | 97.96 | 4 K | **199.00** |
| **LILogicNet-M (ours)** | 98.45 | 8 K | 105.63 |
| **LILogicNet-L (ours)** | 98.95 | 32 K | 27.97 |

(a) MNIST

| Method | Acc [%] | #Gates | ΔAcc/G [$\times 10^{-5}$] |
|---|---|---|---|
| FINN CNV [31] | 80.10 | 901 M | 0.003 |
| DiffLogic Net-S [22] | 51.27 | 48 K | 2.65 |
| DiffLogic Net-M [22] | 57.39 | 512 K | 1.44 |
| DiffLogic Net-L [22] | 60.78 | 1.28 M | 0.84 |
| DiffLogic Net-largest [22] | 62.14 | 5.12 M | 0.24 |
| LogicTreeNet-S [23] | 60.38 | 400 K | 2.60 |
| LogicTreeNet-L [23] | 80.17 | 61 M | 0.05 |
| TTNet-S [3] | 50.10 | 565 K | 0.02 |
| TTNet-L [3] | 70.75 | 189 M | 0.01 |
| RV-LGN [33] | 51.30 | 36 K | 3.61 |
| **LILogicNet-S (ours)** | 55.11 | 8 K | **63.88** |
| **LILogicNet-M (ours)** | 57.66 | 64 K | 11.97 |
| **LILogicNet-L (ours)** | 60.98 | 256 K | 4.29 |

(b) CIFAR-10

Table 4. Accuracy vs. gate count comparison for (a) MNIST and (b) CIFAR-10. ΔAcc/G indicates the accuracy gain per gate: for MNIST over a 90% baseline; for CIFAR-10 over a 50% baseline. LILogicNet models achieve higher ΔAcc/G compared to prior work.

#### 4.3.2. LILogic Net: Accuracy-Efficiency Pareto Front

Based on accuracy, gate count, and training time, we identify three efficient design points. For the MNIST dataset:

- LILogicNet-S: 1L–4K (97.96%, 2.5 min),
- LILogicNet-M: 1L–8K (98.45%, 4.3 min),
- LILogicNet-L: 2Top32–16K (98.95%, 57.1 min).

For the CIFAR-10 dataset:

- LILogicNet-S: 1L–8K (55.11%, 45 min),
- LILogicNet-M: 1L–64K (57.66%, 2 h 20 min),
- LILogicNet-L: 2Top32–128K (60.98%, 4 h 57 min).

These configurations offer strong performance at minimal hardware and time cost, making them ideal candidates for practical deployment.

### 4.4. Comparison with previous work

Table 4 compares LILogicNet with prior logic-based models on MNIST and CIFAR-10. On MNIST, LILogicNet-S achieves 97.96% with 4K gates, outperforming DiffLogic Net-S (97.69%, 48K gates). LILogicNet-M reaches 98.45% with 8K gates, matching larger models such as LogicTreeNet-S (147K) and DiffLogic Net (384K). LILogicNet-L attains 98.95% with 32K gates, approaching LogicTreeNet-M/L (566K/1.27M). RV-LGN and eXpLogic use few gates (18K/5K) but achieve lower accuracy (95.80%, 91.20%), highlighting the challenge of compact high-performance designs. Overall, LILogicNet extends the Pareto frontier, offering competitive accuracy at minimal hardware cost.

For CIFAR-10, all LILogicNet variants—S, M, and L—outperform their corresponding DiffLogic Net baselines (S, M, and L) while using an order of magnitude fewer gates. The largest variant, LILogicNet-L, also surpasses LogicTreeNet-S in accuracy, despite being much smaller. Although the largest DiffLogic Net and LogicTreeNet-L models achieve higher accuracy, they rely on much larger architectures. In this work, we limited our exploration to models of up to 256K gates due to hardware memory constraints, which we plan to address in future extensions to evaluate larger configurations.

## 5. Conclusion

We introduced **LILogic Net**, an efficient logic gate network that jointly learns gate functions and their interconnections. By optimizing the connectome during training and employing a projection-based gate evaluation, our models achieve fast, stable convergence without compromising accuracy. For instance, the 8K-gate LILogicNet-M reaches 98.45% on MNIST in only 4.3 minutes, outperforming prior logic-based models that use up to two orders of magnitude more gates.

Our design favors simplicity over heavy regularization or handcrafted initialization, yet the models generalize strongly, indicating substantial untapped optimization potential. Future extensions—such as dropout, weight decay, alternative scaling, or custom normalization—could further enhance robustness and training efficiency.

Although CIFAR-10 experiments were limited by hardware, they already demonstrate scalability to more complex data. Planned improvements with custom-optimized kernels will enable larger models and deeper architectures.

Overall, LILogic Net lays a strong foundation for efficient, interpretable, and hardware-friendly logic networks, bridging symbolic computation and modern neural design.

## References


[1] Atish Agarwala, Jeffrey Pennington, Yann Dauphin, and Sam Schoenholz. Temperature check: theory and practice for training models with softmax-cross-entropy losses. *arXiv preprint arXiv:2010.07344*, 2020. 4

[2] Colby Banbury, Vijay Janapa Reddi, Paul Torelli, et al. Mlperf tiny benchmark. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. 1

[3] Adrien Benamira, Thomas Peyrin, Trevor Yap, Tristan Guérand, and Bryan Hooi. Truth table net: Scalable, compact & verifiable neural networks with a dual convolutional small boolean circuit networks form. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2024. 8

[4] M. Champs and B. Baldi. An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks. *Future Internet*, 12(7):113, 2023. 1

[5] Ian Clester. Synthesizing music with logic gate networks. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 618–622, 2025. 2

[6] Xue Geng, Zhe Wang, Chunyun Chen, Qing Xu, Kaixin Xu, Chao Jin, et al. From algorithm to hardware: A survey on efficient and safe deployment of deep neural networks. *arXiv preprint arXiv:2405.06038*, 2024. 1

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. 1

[8] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1

[9] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016. 2

[10] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *journal of machine learning research*, 18(187):1–30, 2018. 2

[11] Norman P. Jouppi, Cliff Young, Nishant Patil, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12. ACM, 2017. 1

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015. 3, 4

[13] Fabian Kresse, Emily Yu, Christoph H Lampert, and Thomas A Henzinger. Logic gate neural networks are good for verification. *arXiv preprint arXiv:2505.19932*, 2025. 2

[14] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2, 3

[15] Ian Kuon, Russell Tessier, and Jonathan Rose. Fpga architecture: Survey and challenges. *Foundations and Trends in Electronic Design Automation*, 2(2):135–253, 2007. 1

[16] 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. 1

[17] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. MNIST handwritten digit database. `http://yann.lecun.com/exdb/mnist/`, 1998. Accessed: 2025-05-31. 2, 3

[18] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *European conference on computer vision*, pages 143–159. Springer, 2020. 2

[19] Xiangzhong Luo, Di Liu, Hao Kong, Shuo Huai, Hui Chen, et al. Efficient deep learning infrastructures for embedded computing systems: A comprehensive survey and future envision. *arXiv preprint arXiv:2411.01431*, 2024. 1

[20] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016. 2

[21] Diksha Moolchandani et al. Review of asic accelerators for deep neural network. *Microprocessors & Microsystems*, 89, 2022. 1

[22] Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Deep differentiable logic gate networks. *Advances in Neural Information Processing Systems*, 35:2006–2018, 2022. 1, 2, 3, 4, 8

[23] Felix Petersen, Hilde Kuehne, Christian Borgelt, Julian Welzel, and Stefano Ermon. Convolutional differentiable logic gate networks. *Advances in Neural Information Processing Systems*, 37:121185–121203, 2024. 2, 8

[24] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and backward information retention for accurate binary neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2250–2259, 2020. 2

[25] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016. 2

[26] Charles H Roth Jr and Larry L Kinney. *Fundamentals of Logic Design*. Cengage Learning, 2004. 3

[27] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019. 4

[28] Cristina Silvano, Daniele Ielmini, Fabrizio Ferrandi, Leandro Fiorin, et al. A survey on deep learning hardware accelerators for heterogeneous hpc platforms. *arXiv preprint arXiv:2306.15552*, 2023. 1

[29] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*. Edinburgh, 2003. 3

[30] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017. 1

[31] 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*, pages 65–74, 2017. 2, 8

[32] Emile Van Krieken, Erman Acar, and Frank Van Harmelen. Analyzing differentiable fuzzy logic operators. *Artificial Intelligence*, 302:103602, 2022. 3

[33] Xingbo Wang, Chenxi Feng, Xinyu Kang, Yuru Li, Yucong Huang, and Terry Tao Ye. Logic gate network inference acceleration with risc-v custom instruction set. In *Proceedings of the 22nd ACM International Conference on Computing Frontiers*, pages 205–211, 2025. 2, 8

[34] Stephen Wormald, David Koblah, Matheus Kunzler Maldaner, Domenic Forte, and Damon L Woodard. explogic: Explaining logic types and patterns in difflogic networks. In *International Conference on Information Technology-New Generations*, pages 282–292. Springer, 2025. 2, 8

[35] Shakir Yousefi, Andreas Plesner, Till Aczel, and Roger Wattenhofer. Mind the gap: Removing the discretization gap in differentiable logic gate networks. *arXiv preprint arXiv:2506.07500*, 2025. 2

[36] Chunyu Yuan and Sos S Agaian. A comprehensive review of binary neural network. *Artificial Intelligence Review*, 56(11): 12949–13013, 2023. 2

[37] Chang Yue and Niraj K Jha. Learning interpretable differentiable logic networks. *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, 2024. 2

[38] Chang Yue and Niraj K Jha. Learning interpretable differentiable logic networks for tabular regression. *arXiv preprint arXiv:2505.23615*, 2025. 2

[39] Lotfi A Zadeh. Fuzzy sets. *Information and Control*, 8(3): 338–353, 1965. 3

[40] Xia Zhao, Limin Wang, Yufei Zhang, Xuming Han, Muhammet Deveci, and Milan Parmar. A review of convolutional neural networks in computer vision. *Artificial Intelligence Review*, 57(4):99, 2024. 1