Overcoming Kernel Redundancy for Scaling Logic Gate Networks

Sejin Park **Hongjae Lee** **Changwoo Han** **Seung-Won Jung***

Department of Electrical Engineering
Korea University
{sejin5485, jimmy9704, hcwoo329, swjung83}@korea.ac.kr

Abstract

Differentiable logic gate networks, which operate using only logic gates, have recently attracted attention as an efficient alternative to conventional neural networks. However, despite their efficiency, the scaling behavior of logic gate networks remains underexplored. By contrast, scaling model capacity is a central design principle in deep neural networks and typically leads to improved performance. This discrepancy raises a key question: Can similar scaling benefits also be achieved in logic gate networks? In this work, we focus on width as a primary scaling axis and conduct a systematic analysis of its behavior in logic gate networks. We observe that naive width scaling often introduces redundancy among logic kernels, limiting the effective use of additional kernels and leading to performance saturation. To address this limitation, we propose a dynamic logic kernel framework that reorganizes kernel utilization by promoting specialization across kernel groups. This enables the network to better utilize increased width via input-dependent kernel routing, while ensuring that both routing and computation are implemented entirely with gate-level Boolean operations at inference time. We further find that kernel redundancy is most pronounced at the first gate level, motivating an early-stage dynamic logic kernel strategy that concentrates adaptation at this level. Experimental results demonstrate that our approach improves kernel utilization and increases kernel diversity, leading to higher accuracy with improved parameter efficiency.

1 Introduction

Deep learning has shown remarkable performance across a wide range of tasks; however, its substantial computational cost has driven increasing interest in efficient model design [1, 2]. To address this challenge, various approaches have been proposed to reduce computational cost while maintaining performance. For example, quantization lowers numerical precision to reduce computation while preserving most of the accuracy [3, 4], and binary neural networks further simplify computation by representing both weights and activations with binary values, enabling efficient bit-wise operations [5, 6]. These efficiency-oriented approaches have been successfully applied across diverse domains, including image restoration, language modeling, and other real-world applications [7, 8].

Recently, differentiable logic gate networks have emerged as a promising direction for efficient deep learning, where computations are performed using only logic gate operations instead of conventional arithmetic operations [9]. By leveraging Boolean functions and bit-level representations, these models enable highly efficient and lightweight inference, making them well-suited for resource-constrained environments. Building upon this paradigm, logic gate networks have been extended to structured architectures such as logic tree-based models, where localized tree kernels operate analogously to

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

convolutional layers [10]. In parallel, recent studies have focused on improving the optimization of logic-based models through methods such as continuous relaxation and reparameterization [11, 12].

Despite these advances, existing works primarily focus on improving training strategies or architectural design, with limited attention to how model capacity is structured and utilized under scaling [13, 14]. In particular, while increasing model width introduces multiple logic kernels, it remains unclear whether this additional capacity leads to meaningful functional diversity or merely redundant representations. In contrast to modern deep neural networks, where increased model capacity typically translates into consistent performance gains, such scaling behavior is not yet established in logic gate networks. This raises the question of how multiple kernels are utilized in practice and how they contribute to performance as the model scales.

To answer this question, we conduct a systematic analysis of width scaling in logic gate networks. Our analysis reveals that increasing the number of logic kernels through width scaling leads to redundant kernels, resulting in performance saturation. This finding suggests that the limitation of width scaling does not stem from insufficient capacity, but rather from inefficient utilization of the available kernels.

Based on this insight, we propose Dynamic Logic Kernel (DLK), a logic gate-based routing framework that selects kernel groups according to each input. Rather than using a conventional real-valued gating module, DLK formulates kernel group selection itself as a Boolean routing operation, keeping routing computation within the logic gate domain at inference time. By specializing different kernel groups for different inputs, DLK allows additional width to be used more effectively. We further examine where redundancy appears within the logic kernel hierarchy and find that redundancy is most pronounced at the first gate level. This motivates Early-stage Dynamic Logic Kernel (EDLK), a logic gate network-specific dynamic mechanism that routes first-level gate groups as logic components.

Our contributions are as follows:

- **Scaling analysis.** We present a systematic analysis of width scaling in logic gate networks and reveal that naively increasing the number of logic kernels induces redundancy, resulting in inefficient kernel utilization and performance saturation.
- **Logic kernel specialization.** We propose Dynamic Logic Kernel (DLK), a logic gate-based routing mechanism that reorganizes kernel utilization through input-dependent group selection. This promotes specialization across kernel groups while ensuring that both routing and computation are implemented entirely using gate-level Boolean operations.
- **Early-stage adaptation.** Through level-wise kernel analysis, we identify that redundancy is most pronounced at the first gate level. This observation motivates Early-stage Dynamic Logic Kernel (EDLK), which introduces a logic gate network-specific dynamic mechanism that treats first-level gate groups as selectively routable logic components.
- **Scalable efficiency and performance.** We demonstrate that our approach improves kernel diversity and model-capacity utilization, leading to improved performance under width scaling.

2 Background

2.1 Preliminaries

Differentiable logic gate networks. Logic gate networks replace conventional neurons with Boolean operators such as AND, OR, and XOR, where each neuron selects one of 16 possible Boolean functions. Since Boolean functions are discrete, differentiable logic gate networks use continuous relaxations of logic operators [9]. Following prior work, let $g_1, \ldots, g_{16}$ denote the 16 relaxed binary logic operators. Then, for a trainable parameter vector $\mathbf{z} \in \mathbb{R}^{16}$, a differentiable logic gate $f_{\mathbf{z}}$ is defined as

$$f_{\mathbf{z}}(a_1, a_2) = \sum_{i=1}^{16} \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{16} e^{\mathbf{z}_j}} g_i(a_1, a_2), \tag{1}$$

where $\mathbf{z}_i$ denotes the i-th element of $\mathbf{z}$, and $a_1, a_2 \in [0, 1]$ denote the inputs to the gate. After training, the gate is discretized by selecting the operator with the highest probability.

To improve expressivity, multiple logic gates are composed into a tree structure that serves as a kernel, referred to as a logic tree [10]. A logic kernel applies logic gates hierarchically, where each

internal node takes two inputs and produces one output. For example, a depth-two logic kernel can be represented as $f_{\mathbf{z}_3}\big(f_{\mathbf{z}_1}(a_1, a_2),\ f_{\mathbf{z}_2}(a_3, a_4)\big)$. Such logic kernels are applied over local receptive fields and operate analogously to convolutional filters, where multiple kernels are used in parallel to extract features from different input regions. In this formulation, increasing the number of kernels corresponds to width scaling of the network. For more details about logic gate networks, please refer to prior work [10].

2.2 Related Work

Recent studies have explored several directions for improving logic gate networks, including discrepancy reduction and reparameterization. For example, stochastic relaxations based on Gumbel-Softmax have been introduced to alleviate the discretization gap between continuous training and discrete inference [11]. Another line of work reformulates the parameterization of logic gate networks using truth-table representations [12]. These approaches improve optimization stability and reduce discretization errors through more effective parameterization and relaxation strategies.

In addition, logic gate networks have also been extended to broader application domains. LogicCBM introduces differentiable logic operations into concept-based models to capture non-linear relationships between concepts beyond linear aggregation [15]. Differentiable logic circuits have also been proposed for image compression, replacing conventional neural operations with trainable logic or lookup-based computations for hardware-friendly inference [16]. More recently, LogicIR extends logic gate networks to image restoration, introducing a fully logic-gate architecture for low-level vision tasks such as image denoising, deblocking, and deraining [17].

Despite these advances, a fundamental understanding of logic gate networks remains limited. In particular, the role of width scaling has not been systematically studied, and it remains unclear how logic kernels are utilized within the network. To address this gap, Section 3 systematically analyzes kernel utilization under width scaling and reveals redundancy among logic kernels. Building on this analysis, Section 4 introduces the proposed method in two stages: Dynamic Logic Kernel (DLK) and Early-stage Dynamic Logic Kernel (EDLK), the latter motivated by hierarchical diversity analysis.

3 Scaling Behavior in Logic Gate Networks

In conventional deep neural networks, scaling model capacity through width or depth is generally expected to yield consistent performance improvements [14]. However, such scaling behavior is not directly transferable to logic gate networks. Previous work has shown that simply increasing the depth of differentiable logic gate networks does not necessarily yield consistent performance improvements; instead, deeper models can introduce optimization and discretization challenges, such as vanishing gradients and degraded discretized accuracy [11, 12].

This suggests that scaling strategies in logic gate networks may differ fundamentally from those for conventional neural networks. More importantly, the role of width scaling in logic gate networks remains largely underexplored. This raises the question of whether increasing model width improves performance or encounters limitations similar to those observed in depth scaling.

3.1 Effect of Width Scaling

To investigate this question, we first evaluate the effect of width scaling by increasing the number of logic kernels. We use the same model architecture and training settings, varying only the width setting.

Figure 1 summarizes performance under different widths. Here, "Width" denotes the base width setting—the number of logic kernels in the first convolutional logic layer—from which subsequent convolutional layers are scaled by fixed multipliers; "Parameters" reports the total model parameters.

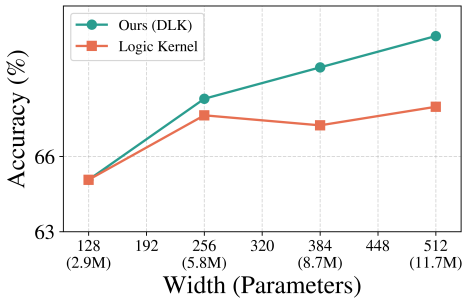


Figure 1: Accuracy under width scaling on the CIFAR-10 dataset.

As shown in Figure 1, the standard "Logic Kernel" baseline exhibits performance saturation, where increasing the number of kernels does not lead

to consistent performance improvements. This observation suggests that simply increasing width is insufficient to effectively utilize the additional capacity. To understand this limitation, we next analyze why naive width scaling fails to improve performance in logic gate networks.

3.2 Diversity of Logic Kernels

Motivated by the above observation, we hypothesize that the performance saturation arises from redundancy among logic kernels. In particular, increasing the number of kernels may cause many of them to produce similar responses rather than capture diverse input patterns. Such redundancy could limit the extent of performance improvement. To examine this hypothesis, we analyze the diversity of logic kernels.

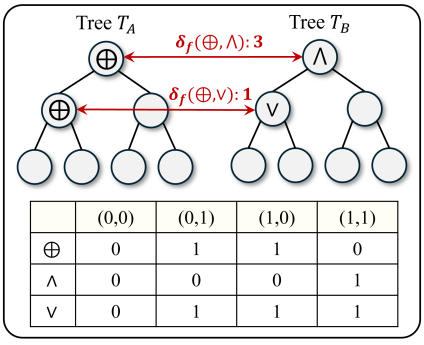
Logic kernel diversity metric. Tree edit distance (TED) has been widely used to measure similarity between tree-structured objects. It quantifies the minimum cost required to transform one tree into another through a sequence of edit operations [18]. In this work, we consider the case where two logic trees share the same topology, under which TED simplifies to a comparison of gate operators at corresponding nodes. Given two trees, T_A and T_B, with identical topology, their TED can be defined as the number of differing gate operators at corresponding nodes:

$$d(T_A, T_B) = \sum_{n=1}^{L} \delta_s\left(g^{T_A(n)}, g^{T_B(n)}\right), \quad \delta_s\left(g^{T_A(n)}, g^{T_B(n)}\right) = \begin{cases} 0, & \text{if } g^{T_A(n)} = g^{T_B(n)}, \\ 1, & \text{otherwise,} \end{cases} \tag{2}$$

where $g^{T_A(n)}$ denotes the gate at the n-th node of tree T_A, and L is the number of nodes.

Each gate in a logic tree corresponds to a Boolean function defined by its outputs over a set of input pairs $\mathcal{A} = \{(0,0),(0,1),(1,0),(1,1)\}$. However, this simplified TED formulation treats all gate mismatches equally and does not account for their functional behavior [18]. It therefore cannot distinguish gates with similar input-output behavior from those with substantially different behavior, limiting its ability to capture meaningful functional differences.

For example, the Boolean function $\oplus$ differs from $\vee$ in only one input case, whereas it differs from $\wedge$ in three input cases. This observation highlights the need for a function-aware distance metric. Accordingly, we define the distance between two gates as



	(0,0)	(0,1)	(1,0)	(1,1)
$\oplus$	0	1	1	0
$\wedge$	0	0	0	1
$\vee$	0	1	1	1

Figure 2: Function-aware distance between two aligned logic gates.

$$\delta_f(g, g') = \sum_{(a_1,a_2)\in\mathcal{A}} \mathbf{1}(g(a_1,a_2) \neq g'(a_1,a_2)), \tag{3}$$

where $\mathbf{1}(\cdot)$ denotes the indicator function, $g(a_1,a_2)$ denotes the output of the Boolean function associated with gate g, and the summation counts the number of mismatched outputs over all input combinations. As illustrated in Figure 2, $\delta_f(\oplus,\vee) = 1$ and $\delta_f(\oplus,\wedge) = 3$. By replacing δ_s with δ_f in Eq. (2), we measure the functional diversity within each convolutional logic layer. Given the set of N logic kernels in a layer, $\mathcal{T} = \{T_1,\ldots,T_N\}$, we compute all pairwise distances and summarize the diversity of the kernel set $\mathcal{T}$, denoted as $\mathcal{D}_\mathcal{T}$, by averaging these distances over all kernel pairs:

$$\mathcal{D}_\mathcal{T} = \frac{1}{\binom{N}{2}} \sum_{i<j} d(T_i, T_j). \tag{4}$$

The final diversity is obtained by averaging $\mathcal{D}_\mathcal{T}$ across all convolutional logic layers.

Analysis of the kernel diversity. Table 1 reports kernel diversity measured using our function-aware TED-based metric, which summarizes truth-table differences between aligned Boolean gates in logic kernels. For the conventional method, diversity remains

Table 1: Kernel diversity across widths.

Method	128	256	384	512
Logic Kernel	3.05	2.50	2.05	1.86
Ours (DLK)	3.05	4.61	4.78	4.89

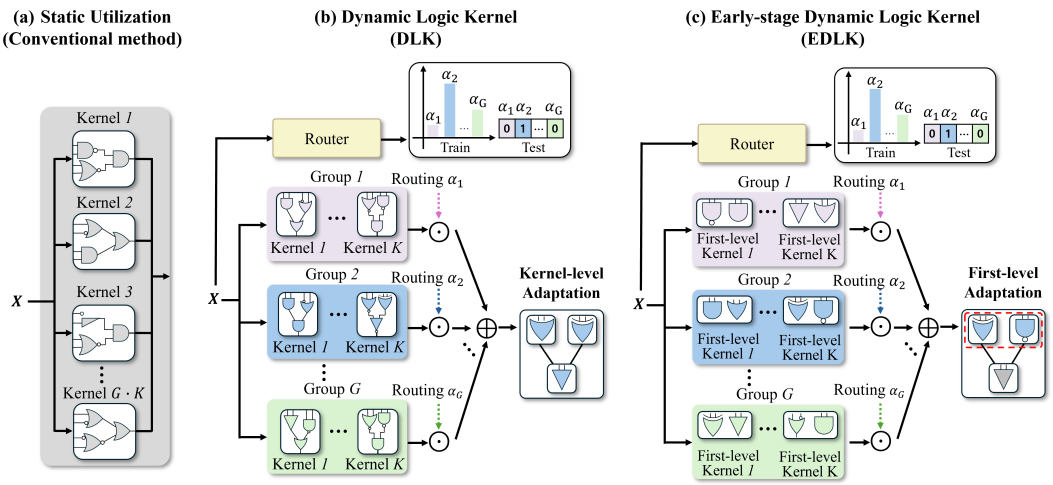


Figure 3: Overview of kernel utilization strategies in logic gate networks: (a) Static utilization, where all kernels are applied without any selection mechanism, (b) dynamic logic kernel (Sec. 4.1), (c) early-stage dynamic logic kernel (Sec. 4.2).

low and further decreases as width increases, indicating that additional kernels become increasingly redundant. This trend is consistent with the performance saturation observed under width scaling.

Overall, these results suggest that naive width scaling tends to introduce redundant kernels rather than translate additional width into effective capacity. This motivates the dynamic logic kernel framework introduced in the next section, which aims to improve kernel utilization under width scaling.

4 Dynamic Kernel Utilization for Scaling Logic Gate Networks

As discussed in Section 3, width scaling in logic gate networks is limited by redundancy among logic kernels rather than by a lack of capacity. This observation shifts the problem from increasing the number of kernels to how they are utilized. In particular, increasing width is most effective when the additional logic kernels produce informative and non-redundant responses. Related ideas have been explored in CNNs through dynamic filter or branch selection, where filters or computational branches are adapted to each input [19–22]. Appendix B provides further discussion of these prior studies and their relation to our logic kernel selection framework.

Therefore, to address this limitation, we propose Dynamic Logic Kernel (DLK), a framework that reorganizes logic kernel utilization through input-dependent selection of logic kernels within a fully logic gate-based formulation. In this work, we treat routing as a logical operation, avoiding real-valued gating functions and continuous mechanisms at inference time. This ensures that the selection process remains fully discrete and consistent with the logic gate formulation. It also allows the additional kernels introduced by width scaling to serve as adaptive candidate groups instead of being applied uniformly. We further examine where redundancy appears within the logic kernel hierarchy. This analysis motivates Early-stage Dynamic Logic Kernel (EDLK), which applies dynamic routing only to first-level gate groups by exploiting the Boolean gate hierarchy of logic kernels.

4.1 Dynamic Logic Kernels

Method. The proposed method is a general module that can be applied to any convolutional logic layer in the network while maintaining a fully logic gate-based formulation. As illustrated in Figure 3(b), the method dynamically routes each input to a selected kernel group, thereby improving kernel utilization. For clarity of explanation, we describe the method using an intermediate convolutional logic layer as an example.

The input feature $\boldsymbol{X}$ is fed to G kernel groups, each consisting of K logic kernels. Let $T_{g,k}(\cdot)$ denote the k-th logic kernel in the g-th group, where $k = 1, 2, \ldots, K$. The g-th kernel group is defined by

concatenating its K kernels as

$$\mathcal{T}_g(\cdot) = \left[T_{g,1}(\cdot)\ ;\ T_{g,2}(\cdot)\ ;\ \cdots\ ;\ T_{g,K}(\cdot)\right], \quad g = 1, 2, \ldots, G, \tag{5}$$

where $[;]$ denotes concatenation of kernels within the group. Accordingly, the output of each group is given by

$$\mathcal{T}_g(\boldsymbol{X}) \in \mathbb{R}^{K \times H_1 \times W_1}. \tag{6}$$

To adapt kernel utilization to each input, we introduce a logic router that produces Boolean routing evidence for each kernel group and selectively activates kernel groups that can extract more informative features from the input. The logic router is implemented by convolutional logic layers followed by global OR-pooling over the spatial dimensions.

Specifically, the router $\mathcal{R}(\cdot)$ takes $\boldsymbol{X}$ as input and produces $\boldsymbol{R} \in [0,1]^{G \times C}$ as follows:

$$\boldsymbol{R} = \mathcal{R}(\boldsymbol{X}), \tag{7}$$

where C denotes the number of bits per group. To obtain a selection score for each group, we aggregate routing bits $\boldsymbol{R}$ along the bit dimension as $s_g = \sum_c \boldsymbol{R}_{g,c}$ for $g = 1, \ldots, G$. This score can be interpreted as the amount of Boolean evidence supporting each kernel group. During training, the routing weight α_g is computed by applying a softmax with temperature τ over the selection scores of all groups:

$$\alpha_g = \frac{e^{s_g/\tau}}{\sum_{g'=1}^{G} e^{s_{g'}/\tau}}. \tag{8}$$

The output feature $\boldsymbol{Y}$ is then given by

$$\boldsymbol{Y} = \sum_{g=1}^{G} \alpha_g \mathcal{T}_g(\boldsymbol{X}). \tag{9}$$

At inference time, the routing scores s_g are computed using a bit-count operation, and we perform hard group selection as

$$g^* = \arg\max_g s_g, \tag{10}$$

and the output is given by

$$\boldsymbol{Y} = \mathcal{T}_{g^*}(\boldsymbol{X}). \tag{11}$$

Effect of dynamic logic kernel. To evaluate DLK under width scaling, we fix the per-group width at $K = 128$ and increase the number of groups G, so the total candidate width grows. As shown in Table 1, DLK maintains high kernel diversity, whereas the conventional baseline exhibits low diversity that further decreases with matched width. Consistently, classification performance also improves with diversity (Figure 1), indicating that enhanced kernel diversity enables more effective utilization of model capacity.

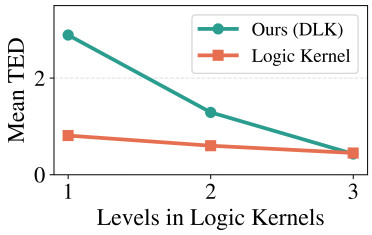


Figure 4: Level-wise decomposition of kernel diversity.

To further investigate where DLK increases kernel diversity relative to the static baseline, we decompose pairwise TED across the logic kernel hierarchy. Specifically, we compute the pairwise mean TED at each gate level. As shown in Figure 4, the diversity gap between DLK and the static Logic Kernel baseline is largest at the first gate level, while later levels show much smaller differences. Since low diversity serves as evidence of kernel redundancy, this suggests that the primary redundancy bottleneck lies at the first gate level. A similar trend is also observed under per-gate normalization in Appendix C.

4.2 Early-stage Dynamic Logic Kernels

Logic gate networks can be viewed as circuits composed of interconnected logic gates. In this view, employing multiple logic kernel groups increases the diversity of available circuit patterns; however, uniformly applying these groups can lead to inefficient use of parameters and hardware resources.

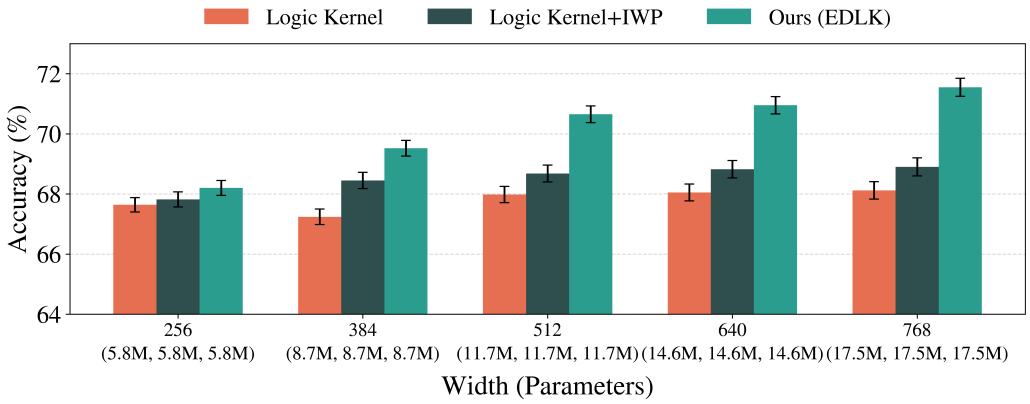


Figure 5: Top-1 accuracy on CIFAR-10 under width scaling with per-group base width $K = 128$.

Since the level-wise analysis indicates that the main bottleneck is most evident at the first gate level, adaptation can be concentrated on this early stage. Based on this insight, we propose an early-stage dynamic logic kernel that introduces level-selective dynamic routing for logic gate networks. Instead of routing entire logic kernels, EDLK dynamically selects only the first-level Boolean gate groups, exploiting the gate-level hierarchy of logic kernels for efficient utilization.

Method. As illustrated in Figure 3(c), we instantiate level-selective routing by selecting only among the first-level components of the logic kernels. In particular, we use the same routing mechanism as in Section 4.1 to compute the selection scores. Specifically, let $T^{(1)}_{g,k}(\cdot)$ denote the first-level components of the k-th kernel in the g-th group. We aggregate the first-level components within each group as

$$\mathcal{T}^{(1)}_g(\cdot) = \left[T^{(1)}_{g,1}(\cdot)\ ;\ \cdots\ ;\ T^{(1)}_{g,K}(\cdot)\right]. \tag{12}$$

The router produces selection scores for these first-level candidates. During training, the adapted first-level output $\boldsymbol{Y}^{(1)}$ is computed as

$$\boldsymbol{Y}^{(1)} = \sum_{g=1}^{G} \alpha_g\, \mathcal{T}^{(1)}_g(\boldsymbol{X}), \tag{13}$$

where α_g is defined analogously to Eq. (8).

The selected first-level output is then passed through the remaining levels of the logic kernel, which are shared across groups. This yields a dynamic kernel layer designed specifically for logic gate networks. It performs input-dependent routing through the explicit Boolean gate hierarchy of logic kernels, rather than by selecting among whole real-valued filters.

5 Experiments

5.1 Experimental Setup

We conduct experiments on CIFAR-10 [23], CIFAR-100 [23], MNIST [24], Flowers-102 [25], SVHN [26], and Tiny-ImageNet [27], with CIFAR-10 as the primary benchmark. We report Top-1 accuracy; for width-scaling experiments, results are averaged over ten random seeds with standard-deviation error bars. All experiments use multi-threshold input binarization following prior work [9] and are conducted on a single NVIDIA RTX 4090 GPU. Details are provided in Appendix A.

We compare the proposed method with logic kernel baselines, including Logic Kernel [10] and Logic Kernel with IWP [12], under the same overall architecture, differing only in kernel utilization. Unless otherwise specified, our method refers to EDLK. At inference, each dynamic layer selects only one group via hard routing, as in Eq. (10). For a fair comparison, reported parameter counts refer to the parameters used at inference time, including all candidate groups and routers.

In the width scaling experiments, "Width" is defined as the total number of logic kernels in the first convolutional logic layer. Static and dynamic models are matched in total width: the static baseline uses $K \cdot G$ kernels, whereas dynamic variants use G groups with K kernels each. For example, in Figure 5, we fix $K = 128$ and vary G, so increasing width increases both the number of groups and the total number of candidate kernels under the matched total width.

5.2 Results and Analysis

Width scaling performance. We evaluate the proposed method on CIFAR-10 under different width settings. We fix the per-group base width to $K = 128$ and scale the total width by increasing the number of groups G under the matched-width protocol. Figure 5 reports Top-1 accuracy averaged over ten seeds, with parameter counts shown alongside the width labels to relate performance trends to model size. As the total width increases, conventional logic kernel-based methods exhibit diminishing returns and tend to saturate, whereas EDLK continues to benefit from additional width. Consequently, the performance gap between EDLK and the static baselines becomes more pronounced at larger widths. These results indicate that simply increasing the number of logic kernels does not necessarily translate into effective model capacity, while input-dependent kernel utilization enables EDLK to make better use of the additional kernels introduced by width scaling.

Table 2: Top-1 accuracy across classification benchmarks. The best scores are in bold.

Method	MNIST	SVHN	CIFAR-100	Flowers-102	Tiny-ImageNet
Logic Kernel	98.55	75.58	58.44	32.40	10.55
Logic Kernel+IWP	98.67	76.05	60.33	33.05	10.86
Ours (EDLK)	**99.10**	**77.00**	**62.46**	**34.26**	**13.35**

Classification benchmark results. Table 2 reports Top-1 accuracy across the classification benchmarks. For a fair comparison, all methods are evaluated at a matched total width of 576, with $(K, G) = (192, 3)$ for the grouped methods, while the number of input bits is set separately for each dataset. Overall, EDLK achieves the best performance across all reported datasets. While the static Logic Kernel and IWP baselines show limited gains at this larger width, EDLK continues to benefit from the increased capacity, indicating more effective utilization of the additional logic kernels.

Table 3: Frozen versus learned routing under the matched-width setting used in Table 2.

Routing strategy	CIFAR-10	Flowers-102
Static, no routing	68.56	32.40
Frozen random	69.35	33.05
Learned (EDLK)	**70.28**	**34.26**

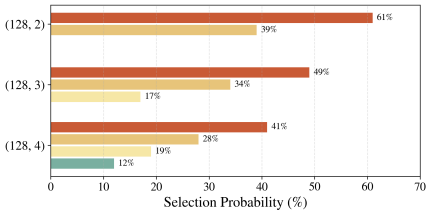


Figure 6: Routing distribution across kernel groups.

Routing behavior. To verify that the performance gain arises from learned input-dependent routing rather than merely introducing multiple kernel groups, we compare EDLK with an identically structured router whose parameters are randomly initialized and frozen throughout training. The candidate kernel groups and the remaining network are trained normally.

As shown in Table 3, frozen random routing provides only marginal improvements over the static baseline, whereas learned routing yields substantially larger gains on both datasets. This indicates that the improvement depends on learning a functionally meaningful routing policy rather than simply introducing random routing among multiple kernel groups.

We further analyze inference-time routing statistics at conv4, aggregated over the CIFAR-10 test set. As shown in Figure 6, routing is neither uniform nor dominated by a single group. Together, these results suggest that the learned router avoids group collapse while enabling input-dependent specialization across kernel groups.

Kernel diversity comparison. We further examine the relationship between kernel diversity and performance by jointly considering the accuracy trend in Figure 1 and the diversity results at total

Table 4: Comparison of kernel diversity across methods at fixed width (CIFAR-10, width 256).

Method	Diversity
Logic Kernel	2.50
Logic Kernel+IWP	3.35
Ours (DLK)	4.61
Ours (EDLK)	4.33

Table 5: Relative reduction in logic-kernel parameters at conv4 and accuracy difference of EDLK compared to DLK.

(K, G)	Reduction (%)	Δ Acc (%)
(128, 2)	21.43	-0.10
(128, 3)	28.57	-0.03
(128, 4)	32.14	-0.15

width 256 in Table 4. Consistent with our findings, the higher kernel diversity of DLK is accompanied by improved accuracy over the static baseline. Table 4 further shows that EDLK achieves substantially higher diversity than the static baselines while remaining comparable to DLK. This suggests that level-selective adaptation to early-stage gates largely preserves the functional diversity induced by dynamic kernel utilization. Together with the observed accuracy trend, these results support the view that reduced kernel redundancy is associated with more effective use of model capacity.

Comparison with DLK. To evaluate the effectiveness of the proposed early-stage adaptation, we compare EDLK with DLK in terms of parameter efficiency and accuracy. Table 5 reports the relative parameter reduction at conv4 and the corresponding Top-1 accuracy difference under matched (K, G) settings. Across different (K, G) settings, EDLK reduces parameters while maintaining accuracy comparable to DLK.

Table 6: Hardware cost and classification performance on MNIST.

Method	Latency (ns)↓	Acc. (%)↑	Params	Energy (pJ)↓	Area (nm^2)↓
Logic Kernel	22.06	95.84	21K	4.15	12.55
Logic Kernel	28.43	96.11	42K	5.34	16.15
Ours (EDLK, $G = 2$)	23.64	**97.29**	22K	4.37	13.21

Hardware efficiency. To evaluate the practical overhead of EDLK beyond total parameter count, we measure FPGA latency on an Intel Cyclone V C9 device and estimate energy consumption and hardware area through synthesis using TSMC N5 technology, following prior hardware evaluation protocols [28]. All measurements are conducted on MNIST with binary inputs. Importantly, the reported hardware costs include the entire EDLK circuit, including all candidate kernel groups, shared upper gate levels, and the complete Boolean router.

As shown in Table 6, EDLK with 22K parameters achieves 97.29% accuracy with 23.64 ns latency, compared with 95.84% and 22.06 ns for the width-matched 21K-parameter Logic Kernel. Despite the additional routing mechanism, the resulting energy and area overhead remains modest. Moreover, compared with the wider 42K-parameter Logic Kernel, EDLK achieves higher accuracy while requiring lower latency, energy, and area. These results indicate that EDLK improves kernel utilization without sacrificing the hardware-efficient characteristics of logic gate networks.

Table 7: Hardware comparison with recent efficient models on MNIST.

Method	BOPs↓	Energy (pJ)↓	Area (nm^2)↓	Acc. (%)↑
TTNet [29]	360K	225	680.75	98.02
DWN [30]	81K	50.63	153.17	98.20
NeuralLUT [31]	75K	46.88	141.82	97.72
Ours (EDLK, $G = 4$)	**11K**	**6.88**	**20.80**	**98.25**

Comparison with efficient models. We further compare EDLK with lightweight variants of recent efficiency-oriented models on MNIST in terms of accuracy and hardware cost. As shown in Table 7, EDLK achieves higher classification accuracy while requiring substantially fewer BOPs, lower energy consumption, and smaller hardware area than the compared methods. These results indicate that EDLK provides a favorable accuracy–efficiency trade-off while retaining the lightweight characteristics of logic gate networks.

Effect of network depth scaling. We further evaluate the effect of increasing network depth under a fixed width setting, $(K, G) = (128, 2)$. As shown in Figure 7, EDLK consistently outperforms the static baselines across different depth settings. However, increasing the depth does not yield additional performance gains; instead, accuracy gradually saturates or slightly decreases for all methods. This suggests that the proposed dynamic kernel utilization primarily addresses the limitations of width scaling, while the optimization and discretization challenges associated with deeper logic gate networks remain [11].

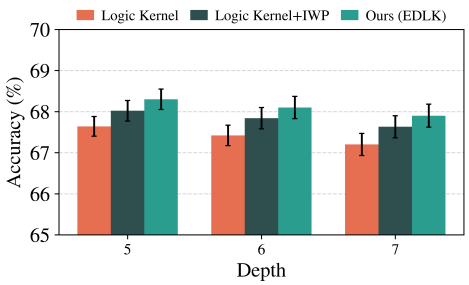


Figure 7: Top-1 accuracy under network depth scaling on CIFAR-10 with fixed width.

6 Limitations

Our study has several limitations. First, although our method improves kernel utilization under width scaling, extending the gate-based formulation to substantially larger-scale architectures, such as LLM-scale models, remains challenging. Second, our evaluation mainly focuses on image classification, leaving dense prediction tasks such as image restoration, semantic segmentation, and optical flow unexplored. Finally, logic gate networks remain an emerging paradigm and still lag behind mature architectures such as CNNs and Transformers in terms of predictive accuracy. Nevertheless, their discrete computation and hardware-friendly inference characteristics make them a promising direction for efficient model design.

7 Future Work

Kernel redundancy may arise from three complementary limitations in current logic gate networks: (1) **optimization**, where unstable or imbalanced training can cause different kernels to converge to similar Boolean functions; (2) **architecture**, where sparse and local information propagation may limit effective receptive-field utilization; and (3) **representation**, where fixed binary input encoding may restrict the information available to subsequent logic computations. Future work could therefore investigate more stable training strategies, improved input representations, and new architectural designs that better exploit the spatial structure of visual data, toward more scalable differentiable logic gate networks for vision tasks.

8 Conclusion

In this work, we revisit width scaling in logic gate networks, a fundamental yet underexplored aspect of model scaling. While scaling has been a central principle in deep learning, its behavior in logic gate networks has not been systematically understood. Through a comprehensive analysis, we reveal that naive width expansion leads to kernel redundancy, resulting in performance saturation rather than consistent gains. We further analyze redundancy across the logic kernel hierarchy and find that the first gate level is the primary source of redundancy. Based on this insight, we propose a dynamic logic kernel framework to better organize model capacity and introduce an early-stage adaptation strategy specific to logic gate networks. By formulating group selection as Boolean routing, our framework avoids external real-valued gating and preserves a fully logic gate-based inference path. Experimental results on standard benchmarks demonstrate that the proposed method improves accuracy while promoting more effective utilization of logic kernels. Overall, our study highlights that the limitation of scaling in logic gate networks is not merely a matter of insufficient capacity, but a more fundamental issue in how logical functions are structured and utilized.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2025-16068196), and National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2026-25608457).

References

[1] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[2] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning*, pages 6105–6114, 2019.

[3] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[4] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[5] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in Neural Information Processing Systems*, 29, 2016.

[6] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision*, pages 525–542, 2016.

[7] Jing Liu, Ruihao Gong, Xiuying Wei, Zhiwei Dong, Jianfei Cai, and Bohan Zhuang. QLLM: Accurate and efficient low-bitwidth quantization for large language models. In *Proceedings of the International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=FIplmUWdm3`.

[8] Sejin Park, Sangmin Lee, Kyong Hwan Jin, and Seung-Won Jung. IM-LUT: Interpolation mixing look-up tables for image super-resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14317–14325, 2025.

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

[10] 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.

[11] Shakir Yousefi, Andreas Plesner, Till Aczel, and Roger Wattenhofer. Mind the gap: Removing the discretization gap in differentiable logic gate networks. *Advances in Neural Information Processing Systems*, 38, 2025.

[12] Lukas Rüttgers, Till Aczel, Andreas Plesner, and Roger Wattenhofer. Light differentiable logic gate networks. In *Proceedings of the International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=EaGQ5luZtf`.

[13] Irwan Bello, William Fedus, Xianzhi Du, Ekin Dogus Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting resnets: Improved training and scaling strategies. *Advances in Neural Information Processing Systems*, 34:22614–22627, 2021.

[14] Behrooz Ghorbani, Orhan Firat, Markus Freitag, Ankur Bapna, Maxim Krikun, Xavier Garcia, Ciprian Chelba, and Colin Cherry. Scaling laws for neural machine translation. In *Proceedings of the International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=hR_SMu8cxCV`.

[15] Deepika SN Vemuri, Gautham Bellamkonda, Aditya Pola, and Vineeth N Balasubramanian. LogicCBMs: Logic-enhanced concept-based learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 6039–6048, 2026.

[16] Till Aczel, David F Jenny, Simon Jonas Bührer, Andreas Plesner, Antonio Di Maio, and Roger Wattenhofer. IC-DLC: Differentiable logic circuits for hardware-friendly image compression. In *Proceedings of the AAAI Workshop on Machine Learning for Wireless Communication and Networks*, 2026.

[17] Hongjae Lee, Myungjun Son, Jaeseong Yu, and Seung-Won Jung. Logicir: Logic gate networks for image restoration. In *Proceedings of the European Conference on Computer Vision*, pages 394–411, 2026.

[18] Kaizhong Zhang, Rick Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42(3):133–139, 1992.

[19] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. *Advances in Neural Information Processing Systems*, 29, 2016.

[20] Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. Selective kernel networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 510–519, 2019.

[21] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. *Advances in Neural Information Processing Systems*, 32, 2019.

[22] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11030–11039, 2020.

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

[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[25] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Proceedings of the Indian Conference on Computer Vision, Graphics & Image processing*, pages 722–729, 2008.

[26] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*. Granada, 2011.

[27] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[28] Marina Neseem, Conor McCullough, Randy Hsin, Chas Leichner, Shan Li, In Suk Chong, Andrew Howard, Lukasz Lew, Sherief Reda, Ville-Mikko Rautio, et al. Pikelpn: Mitigating overlooked inefficiencies of low-precision neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15996–16005, 2024.

[29] Adrien Benamira, Tristan GuÃŠrand, Thomas Peyrin, Trevor Yap, and Bryan Hooi. A scalable, interpretable, verifiable & differentiable logic gate convolutional neural network architecture from truth tables. *arXiv preprint arXiv:2208.08609*, 2022.

[30] 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, editors, *Proceedings of the International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 2277–2295. PMLR, 21–27 Jul 2024. URL `https://proceedings.mlr.press/v235/bacellar24a.html`.

[31] 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)*, pages 208–216. IEEE, 2025.

[32] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23 (120):1–39, 2022.

[33] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 764–773, 2017.

[34] Ningning Ma, Xiangyu Zhang, Jiawei Huang, and Jian Sun. Weightnet: Revisiting the design space of weight networks. In *Proceedings of the European Conference on Computer Vision*, pages 776–792, 2020.

[35] Duo Li, Jie Hu, Changhu Wang, Xiangtai Li, Qi She, Lei Zhu, Tong Zhang, and Qifeng Chen. Involution: Inverting the inherence of convolution for visual recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12321–12330, 2021.

[36] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7132–7141, 2018.

[37] Weizhe Hua, Yuan Zhou, Christopher M De Sa, Zhiru Zhang, and G Edward Suh. Channel gating neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[38] Zhuo Su, Linpu Fang, Wenxiong Kang, Dewen Hu, Matti Pietikäinen, and Li Liu. Dynamic group convolution for accelerating convolutional neural networks. In *Proceedings of the European Conference on Computer Vision*, pages 138–155, 2020.

A Experimental details and additional results

A.1 Network architectures

We use a common design principle across datasets and adapt the convolutional logic backbone according to the input resolution and input bit-depth [10, 12]. For example, 32×32 RGB datasets use the same four-stage backbone schedule, 64×64 RGB datasets use the same five-stage schedule, and MNIST uses a separate three-stage grayscale backbone. Throughout Appendix A, we follow the width convention used in Section 5.1.

Width notation. Specifically, K denotes the base number of logic kernels in each candidate group, and G denotes the number of candidate groups instantiated in a dynamic convolutional logic layer. The matched width scale $W = KG$ is defined as the candidate-kernel width of the first convolutional stage, i.e., the number of logic kernels in conv1 under the matched-width notation. Later convolutional stages are specified by fixed multipliers of this first-stage width: a stage with multiplier m contains $mW = mKG$ candidate logic kernels before routing. These kernels are organized as G groups of mK kernels for dynamic variants, or as mW kernels for static baselines. Unless otherwise stated, all convolutional stage widths below are expressed relative to this first-stage matched width W.

Static and dynamic convolutional stages. Static logic kernel baselines, including Logic Kernel and Logic Kernel with IWP [10, 12], implement each W-scaled convolutional stage as a standard logic kernel layer, without input-dependent routing. Dynamic variants retain the same W-scaled convolutional stages, but partition their kernels into candidate groups routed on a per-input basis. The router combines the candidate-group outputs along the group dimension, rather than concatenating them into additional downstream channels.

Parameter-budget matching. Our main comparison is designed to evaluate how different kernel-utilization mechanisms perform under a comparable overall parameter budget. To avoid method-specific changes in downstream classifier capacity, we keep the subsequent standard Logic Layers fixed under the same W-scaled setting used by the existing baselines. Thus, the convolutional layers differ in how kernels are utilized, while the classifier head uses the same structure across methods. We report the total inference-time parameter count for each model, including all candidate kernel groups in the dynamic variants, the dynamic routers, and the subsequent standard Logic Layers.

Dynamic-stage selection. Across datasets, we instantiate dynamic logic kernels only in the highest-width convolutional stages, while keeping the rest of the convolutional logic backbone unchanged. This placement reflects the goal of DLK/EDLK: rather than redesigning the entire network, we attach dynamic routing to the stages where the candidate-kernel budget is largest under width scaling. The earlier stages therefore remain standard convolutional logic layers, and the proposed mechanism is introduced as a stage-level plug-in to the existing backbone. This allows the dynamic variants to largely retain the original architecture, while improving the utilization of widened logic kernels through only a small number of high-width stage replacements.

A.1.1 Tiny-ImageNet

For Tiny-ImageNet, which consists of 64×64 RGB images, we use a five-stage convolutional logic backbone. Each image is resized to 64×64 and converted into a binary representation using channel-wise multi-threshold thermometer encoding. In the reported setting, we use $C_b = 127$ thresholds per RGB channel, yielding $3C_b = 381$ binary input channels. The five convolutional stages are configured with widths W, $4W$, $16W$, $32W$, and $32W$, respectively. Each stage uses a logic tree kernel of depth $d = 3$ with a 3×3 receptive field and padding of one, followed by 2×2 OR-pooling with stride 2.

In the proposed dynamic variants, conv4 and conv5 are replaced with dynamic logic kernel layers, while conv1–conv3 remain standard convolutional logic layers. The static logic kernel baseline uses the same W-scaled convolutional stage structure, but implements all five stages as standard convolutional logic layers. The IWP baseline preserves the same overall topology and replaces the convolutional and fully connected logic layers with their IWP counterparts.

A.1.2 Flowers-102

For Flowers-102 [25], which contains variable-resolution RGB images, we use a five-stage convolutional logic backbone evaluated at 64×64 resolution. Each image is resized and cropped to 64×64 before being converted into a binary representation using thermometer encoding. We use $C_b = 5$ thresholds per RGB channel, yielding $3C_b = 15$ binary input channels. The five convolutional stages are configured with widths W, $4W$, $16W$, $32W$, and $32W$, respectively. Each stage uses a logic tree kernel of depth $d = 3$ with a 3×3 receptive field and padding of one, followed by 2×2 OR-pooling with stride 2.

In the proposed dynamic variants, conv4 and conv5 are replaced with dynamic logic kernel layers, while conv1–conv3 remain standard convolutional logic layers. The static logic kernel baseline uses the same W-scaled convolutional stage structure, but implements all five stages as standard convolutional logic layers. The IWP baseline preserves the same overall topology and replaces the convolutional and fully connected logic layers with their IWP counterparts.

A.1.3 CIFAR-100

For CIFAR-100, which consists of 32×32 RGB images, we use a four-stage convolutional logic backbone. Each image is converted into a binary representation using channel-wise multi-threshold thermometer encoding. We use $C_b = 63$ thresholds per RGB channel, yielding $3C_b = 189$ binary input channels. The four convolutional stages are configured with widths W, $4W$, $16W$, and $32W$, respectively. Each stage uses a logic tree kernel of depth $d = 3$ with a 3×3 receptive field and padding of one, followed by 2×2 OR-pooling with stride 2. Following [10], we additionally incorporate fixed low-level feature detectors to enrich the binary input representation with local spatial information.

In the proposed dynamic variants, conv4 is replaced with a dynamic logic kernel layer, while conv1–conv3 remain standard convolutional logic layers. The static logic kernel baseline uses the same W-scaled convolutional stage structure, but implements all four stages as standard convolutional logic layers. The IWP baseline preserves the same overall topology and replaces the convolutional and fully connected logic layers with their IWP counterparts.

A.1.4 CIFAR-10

For CIFAR-10, which consists of 32×32 RGB images, we use a four-stage convolutional logic backbone. Each image is converted into a binary representation using channel-wise multi-threshold thermometer encoding. We use $C_b = 31$ thresholds per RGB channel, yielding $3C_b = 93$ binary input channels. The four convolutional stages are configured with widths W, $4W$, $16W$, and $32W$, respectively. Each stage uses a logic tree kernel of depth $d = 3$ with a 3×3 receptive field and padding of one, followed by 2×2 OR-pooling with stride 2.

In the proposed dynamic variants, conv4 is replaced with a dynamic logic kernel layer, while conv1–conv3 remain standard convolutional logic layers. The static logic kernel baseline uses the same W-scaled convolutional stage structure, but implements all four stages as standard convolutional logic layers. The IWP baseline preserves the same overall topology and replaces the convolutional and fully connected logic layers with their IWP counterparts.

A.1.5 SVHN

For SVHN [26], which consists of 32×32 RGB images, we use the same four-stage convolutional logic backbone as in CIFAR-10. Each image is converted into a binary representation using $C_b = 2$ thresholds per RGB channel, yielding $3C_b = 6$ binary input channels. The four convolutional stages are configured with widths W, $4W$, $16W$, and $32W$, respectively. Each stage uses a logic tree kernel of depth $d = 3$ with a 3×3 receptive field and padding of one, followed by 2×2 OR-pooling with stride 2.

In the proposed dynamic variants, conv4 is replaced with a dynamic logic kernel layer, while conv1–conv3 remain standard convolutional logic layers. The static logic kernel baseline uses the same W-scaled convolutional stage structure, but implements all four stages as standard convolutional logic layers. The IWP baseline preserves the same overall topology and replaces the convolutional and fully connected logic layers with their IWP counterparts.

A.1.6 MNIST

For MNIST, which consists of 28×28 grayscale images, we use a shallower three-stage convolutional logic backbone. Each image is converted into a binary representation using threshold-based thermometer encoding. We use $C_b = 1$ threshold for the grayscale channel, yielding one binary input channel. The first convolutional stage uses width W with a 3×3 receptive field and zero padding, followed by 2×2 OR-pooling with stride 2. The next two convolutional stages use widths $3W$ and $9W$, respectively, both with 3×3 receptive fields, padding of one, and the same OR-pooling operation.

In the proposed dynamic variant, only the final convolutional stage is replaced with a dynamic logic kernel layer, while the first two stages remain standard convolutional logic layers. The static logic kernel baseline uses the same W-scaled convolutional stage structure, but implements all three stages as standard convolutional logic layers. The IWP baseline preserves the same overall topology and replaces the convolutional and fully connected logic layers with their IWP counterparts.

A.2 Training details

A.2.1 Objective function

We train all models with a supervised classification objective. For DLK and EDLK, we additionally use a routing load-balancing loss for the layers that contain dynamic logic kernels. Consider a mini-batch of size B, indexed by $n = 1, \ldots, B$, with labels $y^{(n)} \in \{1, \ldots, C_{\mathrm{cls}}\}$. Let $\boldsymbol{z}^{(n)} \in \mathbb{R}^{C_{\mathrm{cls}}}$ denote the class logits after the final GroupSum layer.

Cross-entropy loss. The main training loss is the standard cross-entropy loss:

$$\mathcal{L}_{\mathrm{CE}} = -\frac{1}{B} \sum_{n=1}^{B} \log \frac{\exp\left(z_{y^{(n)}}^{(n)}\right)}{\sum_{c=1}^{C_{\mathrm{cls}}} \exp\left(z_c^{(n)}\right)}. \tag{14}$$

This loss supervises the final class logits. In the implementation, it is computed by applying `CrossEntropyLoss` to the output logits and labels.

Layer-wise load-balancing loss. For DLK and EDLK, we adopt the auxiliary load-balancing loss from Switch Transformers [32] to discourage routing collapse. The loss is applied to the routed kernel groups in each dynamic layer and averaged across dynamic layers.

Overall objective. The final training objective is

$$\mathcal{L} = \begin{cases} \mathcal{L}_{\mathrm{CE}}, & \text{static logic kernel baselines,} \\ \mathcal{L}_{\mathrm{CE}} + \lambda_{\mathrm{bal}} \mathcal{L}_{\mathrm{bal}}, & \text{DLK and EDLK.} \end{cases} \tag{15}$$

Here, λ_{bal} controls the strength of the load-balancing regularization.

A.2.2 Optimization and hyperparameters

Common optimization protocol. Unless otherwise stated, the reported experiments are trained with AdamW for $200{,}000$ iterations. The default batch size is 64 with an initial learning rate of 0.02. When the batch size is changed in controlled comparisons, we scale the learning rate linearly with the batch size. For the dynamic variants, the routing bit-width is selected from $[15, 31]$ according to the number of candidate groups G in each layer. All experiments are conducted on a single NVIDIA RTX 4090 GPU.

A.3 Existing assets

We use CIFAR-10 [23], CIFAR-100 [23], MNIST [24], Flowers-102 [25], SVHN [26], and Tiny-ImageNet [27] only as public benchmark datasets for academic evaluation. We cite their original sources, follow the official licenses or terms of use where provided, and do not redistribute, repackage, or modify the datasets.

Table 8: Representative CNN methods related to adaptive kernel or capacity utilization.

Group	Representative methods	Adapted object
Kernel generation and spatial adaptation	Dynamic Filter Networks [19] Deformable ConvNets [33] WeightNet [34] Involution [35]	Filters, weights, sampling support, or spatial kernels
Kernel, branch, and expert selection	Selective Kernel Networks [20] CondConv [21] Dynamic Convolution [22]	Kernels, branches, or experts
Channel and group routing	Squeeze-and-Excitation Networks [36] Channel Gating Neural Networks [37] Dynamic Group Convolution [38]	Channels or groups

B CNN precedents for dynamic kernel utilization

Motivation. Our methods are designed for logic gate networks, but motivation is related to a broader direction in CNNs: rather than applying the same fixed convolutional computation to every input, many methods adapt filters, branches, experts, channels, or groups according to the input. This appendix clarifies the relationship between DLK/EDLK and prior CNN-based approaches to input-dependent kernel utilization. While input-dependent kernel utilization has been explored in CNNs, to the best of our knowledge, this work is the first to formulate and validate this principle in logic gate networks, where both routing and computation are implemented through discretizable Boolean logic.

Representative CNN methods. Table 8 gives a compact grouping of representative CNN methods. To avoid overloading the table, we only list the representative methods and the main adaptive object in each group; the detailed comparison is provided in the following paragraphs.

Kernel generation and spatial adaptation. The first group modifies the convolutional operator itself. Dynamic Filter Networks generate filters conditioned on the input, showing that convolutional filters do not need to be fixed after training [19]. Deformable ConvNets make the spatial support of convolution adaptive by learning offsets for sampling locations [33]. WeightNet studies convolutional weight generation through a weight-generating network, providing a related perspective on how convolutional weights can be parameterized beyond a directly stored static tensor [34]. Involution further changes the standard convolutional design by using spatial-specific operators [35]. These methods show that the effective convolutional operation can be made more flexible than a single fixed filter bank.

Kernel, branch, and expert selection. The second group is the closest CNN analogue to DLK. Selective Kernel Networks use multiple convolutional branches with different receptive-field sizes and adaptively combine their outputs through attention [20]. CondConv learns example-dependent convolutional kernels by combining multiple expert kernels with input-conditioned routing weights [21]. Dynamic Convolution similarly aggregates multiple convolution kernels using attention over kernels [22]. These methods share the idea that a layer can contain multiple candidate transformations and adaptively select or mix them for each input.

Channel and group routing. The third group adapts computation at the channel or group level. Squeeze-and-Excitation Networks recalibrate channel-wise feature responses by modeling dependencies between channels [36]. Channel Gating Neural Networks dynamically skip subsets of channel computations to reduce unnecessary computation [37]. Dynamic Group Convolution selects input channels for each group conditioned on the input, improving the utilization of grouped convolution [38]. These methods are not direct dynamic kernel methods in the same sense as CondConv or Dynamic Convolution. However, they support the broader view that model capacity can be used more effectively when channels or groups are adaptively emphasized rather than uniformly used.

DLK formulation. DLK follows the general principle of input-dependent kernel utilization, but reformulates it for the computational structure of logic gate networks. Unlike CNN-based dynamic kernel methods, where the adapted object is typically a real-valued convolutional filter, DLK treats each kernel as a discretizable Boolean logic tree. It constructs G candidate groups of logic kernels, $\{\mathcal{T}_g\}_{g=1}^{G}$, and uses a logic-gate-based router $\mathcal{R}(\cdot)$ to produce input-dependent group scores. During training, the groups are combined through the soft routing weights in Eq. (8), while inference replaces this relaxation with hard selection as in Eq. (10). Thus, DLK performs conditional selection over Boolean logic-kernel groups rather than real-valued filter modulation, keeping both routing and computation within the logic-gate-based inference formulation.

Selection granularity. A key design choice is the level at which dynamic selection is applied. Whole-layer selection is too coarse because all kernels share the same routing decision, whereas independent gate-wise routing is too fine-grained and would weaken the shared hierarchical structure of a logic kernel. DLK therefore performs selection at the structured kernel group level, allowing different Boolean kernel groups to specialize to different input patterns while preserving a simple logic-compatible routing structure.

This granularity also motivates EDLK. A logic kernel is a hierarchical Boolean tree, so dynamic selection can be applied inside the kernel rather than only to the entire kernel. The level-wise TED analysis in Figure 4 shows that the diversity gap between DLK and the static Logic Kernel baseline is largest at the input-adjacent first gate level, suggesting that the main redundancy bottleneck appears where elementary Boolean primitives are formed. Based on this observation, EDLK localizes dynamic selection to first-level gate groups and shares the remaining upper levels across groups.

EDLK formulation. EDLK applies the same routing principle as DLK, but adapts the selected units to first-level Boolean gate groups. Specifically, it constructs G candidate first-level operators $\{\mathcal{T}_g^{(1)}\}_{g=1}^{G}$ and computes the routing weights $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_G]$ using the router in Section 4.1. During training, the adapted first-level output is computed as

$$\boldsymbol{Y}^{(1)} = \sum_{g=1}^{G} \alpha_g \mathcal{T}_g^{(1)}(\boldsymbol{X}), \tag{16}$$

and the remaining logic-tree levels are shared across groups. At inference time, the soft aggregation is replaced by hard selection of a single first-level gate group, following the same soft-to-hard routing rule as DLK.

This design is specific to logic gate networks. Since the first level forms elementary Boolean interactions from the input activations, selecting among first-level gate groups changes the Boolean primitives supplied to the shared upper-level circuit. EDLK therefore introduces input-dependent variation where Boolean primitives are formed, while preserving the compact and compositional structure of the deeper logic gates.

Logic gate network-specific dynamic routing. DLK and EDLK formulate dynamic routing for a setting that does not naturally arise in ordinary real-valued kernels. CNN-based dynamic-kernel methods typically adapt filters, branches, channels, or experts at the tensor level. In contrast, a logic kernel is a discretizable Boolean circuit composed of explicit gate levels, where each internal node represents a finite two-input Boolean function. This structure enables a routing mechanism that is specific to logic gate networks: DLK selects among Boolean kernel groups, while EDLK selects among structured first-level gate groups inside the kernel hierarchy.

This design highlights why EDLK is specific to logic gate networks rather than a generic compression of dynamic kernels. In ordinary real-valued kernels, there is no explicit hierarchy of discrete Boolean gates where routing can be localized; adaptation is therefore typically defined over filters, channels, branches, or experts. A logic kernel, however, is a gate-level Boolean circuit, so its input-adjacent gate groups form a meaningful and routable unit: they determine the elementary Boolean primitives that are later composed by the shared upper gates. EDLK exploits exactly this logic-specific structure by applying dynamic selection only where Boolean primitives are formed, while keeping the remaining circuit shared. Thus, EDLK introduces a dynamic mechanism that is both needed for addressing first-level redundancy in logic kernels and implementable only because logic gate networks expose an explicit Boolean gate hierarchy.

Table 9: Truth-table representation of the 16 two-input Boolean operators.

$\mathbf{b}(g)$	Boolean operator $g(a_1, a_2)$	00	01	10	11
0000	0	0	0	0	0
0001	$a_1 \wedge a_2$	0	0	0	1
0010	$a_1 \wedge \neg a_2$	0	0	1	0
0011	a_1	0	0	1	1
0100	$\neg a_1 \wedge a_2$	0	1	0	0
0101	a_2	0	1	0	1
0110	$a_1 \oplus a_2$	0	1	1	0
0111	$a_1 \vee a_2$	0	1	1	1
1000	$\neg(a_1 \vee a_2)$	1	0	0	0
1001	$\neg(a_1 \oplus a_2)$	1	0	0	1
1010	$\neg a_2$	1	0	1	0
1011	$a_1 \vee \neg a_2$	1	0	1	1
1100	$\neg a_1$	1	1	0	0
1101	$\neg a_1 \vee a_2$	1	1	0	1
1110	$\neg(a_1 \wedge a_2)$	1	1	1	0
1111	1	1	1	1	1

C Tree Edit Distance

This appendix provides additional details on the function-aware tree edit distance (TED) used for the kernel diversity analysis in Sections 3.2 and 4.1. Since logic kernels within the same convolutional logic layer share an identical tree topology, two kernels can be compared by aligning their internal gate nodes. Let T_A and T_B denote two logic kernels with L internal gates, and let $g^{T_A(n)}$ and $g^{T_B(n)}$ be the Boolean operators assigned to the n-th aligned node of each kernel. The function-aware TED is defined as

$$d(T_A, T_B) = \sum_{n=1}^{L} \delta_f\left(g^{T_A(n)}, g^{T_B(n)}\right). \tag{17}$$

Truth-table representation. Each gate in a logic tree corresponds to a Boolean function defined by its outputs over the input-pair set $\mathcal{A} = \{(0,0), (0,1), (1,0), (1,1)\}$. We therefore represent a gate g using the truth vector

$$\mathbf{b}(g) = \big[g(0,0),\, g(0,1),\, g(1,0),\, g(1,1)\big] \in \{0,1\}^4. \tag{18}$$

Table 9 summarizes the 16 possible two-input Boolean operators under this fixed ordering.

Using this truth-vector representation, the node-wise functional cost is computed as

$$\delta_f(g, g') = \sum_{(a_1,a_2)\in\mathcal{A}} \mathbf{1}(g(a_1,a_2) \neq g'(a_1,a_2)) = \|\mathbf{b}(g) - \mathbf{b}(g')\|_0. \tag{19}$$

This cost counts the number of input cases for which two gates produce different outputs. For example, $\oplus$ and $\vee$ differ only at $(1,1)$, yielding $\delta_f(\oplus, \vee) = 1$, whereas $\oplus$ and $\wedge$ differ in three input cases, yielding $\delta_f(\oplus, \wedge) = 3$. Hence, unlike a binary same-or-different gate comparison, the proposed cost captures the degree of functional discrepancy between Boolean operators.

Kernel diversity and level-wise analysis. Given a set of N logic kernels in a convolutional logic layer, $\mathcal{T} = \{T_1, \ldots, T_N\}$, we quantify kernel diversity using the mean pairwise distance

$$\mathcal{D}_{\mathcal{T}} = \frac{1}{\binom{N}{2}} \sum_{i<j} d(T_i, T_j). \tag{20}$$

The reported diversity score is obtained by averaging this quantity across the convolutional logic layers included in the analysis.

For the level-wise analysis in Figure 4, we compute the same distance while restricting the summation to gates at a specific tree level. We index levels from the input-adjacent gate level to the output-adjacent root level. Let $\mathcal{N}_\ell$ denote the set of aligned nodes at level ℓ. The level-wise distance is

defined as

$$d^{(\ell)}(T_A, T_B) = \sum_{n \in \mathcal{N}_\ell} \delta_f \left(g^{T_A(n)}, g^{T_B(n)} \right). \tag{21}$$

The corresponding level-wise mean pairwise diversity is

$$\mathcal{D}_{\mathcal{T}}^{(\ell)} = \frac{1}{\binom{N}{2}} \sum_{i<j} d^{(\ell)}(T_i, T_j). \tag{22}$$

This quantity gives an additive decomposition of the full-kernel TED across the hierarchy of the logic kernel. In our depth-3 logic kernels, the three gate levels contain different numbers of gates. Therefore, $\mathcal{D}_{\mathcal{T}}^{(\ell)}$ should be interpreted as the additive contribution of level ℓ to full-kernel diversity.

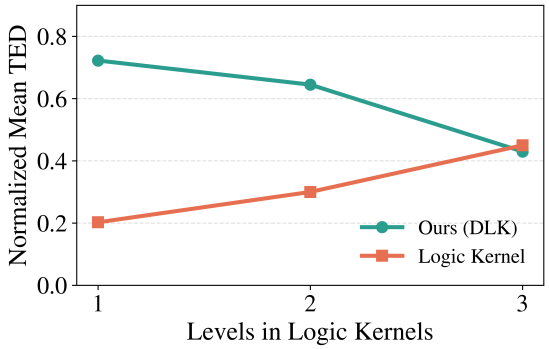


Figure 8: Per-gate normalized level-wise TED on CIFAR-10. Each level-wise TED value is divided by the number of gates at that level. A similar first-level trend is observed after normalization.

TED-based empirical results. Using the discretized gates obtained after training, we compute the mean pairwise function-aware TED between logic kernels within each convolutional logic layer. These layer-wise distances are then averaged across the analyzed layers to obtain the reported diversity score. For the level-wise analysis, the same computation is performed independently at each gate level, enabling a direct comparison of functional diversity across different depths of the logic tree. For the normalized analysis in Figure 8, we divide each level-wise distance by the number of gates at the corresponding level before averaging over kernel pairs.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims in the abstract and Introduction are limited to width scaling in logic gate networks, kernel redundancy, and the proposed DLK/EDLK mechanisms. These claims are supported by the scaling and diversity analysis in Section 3, the proposed method and additional analysis in Section 4, and the empirical results in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6 explicitly discusses the main limitations of the work, including the scalability constraints of the broader logic gate network paradigm, the focus on image classification benchmarks, the lack of validation on dense prediction tasks, and the remaining performance gap compared with mature neural architectures such as CNNs and Transformers.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present formal theoretical results such as theorems or lemmas. The equations in Sections 3 and 4 define the diversity analysis and the proposed DLK/EDLK mechanisms rather than establishing formal theoretical guarantees.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 4 defines the proposed DLK and EDLK mechanisms, including the routing scores, training-time soft routing, and inference-time hard group selection. Section 5.1 specifies the evaluated datasets, Top-1 accuracy metric, input binarization, hard-routing inference protocol, parameter-counting rule, shared architecture comparison, hardware, and random-seed averaging used for the main experiments. Moreover, we provide more details in Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: All datasets used in our experiments are publicly available and cited in Section 5.1. The initial submission does not include a public code release, but Sections 4, 5.1, and Appendix A provide the method, training setup, and reproduction details; we will release the code and reproduction instructions.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5.1 specifies the evaluated datasets, primary benchmark, Top-1 accuracy metric, inference-time hard routing, parameter-counting protocol, input binarization, shared architecture comparison, hardware, and random-seed averaging. The accompanying reproduction material fixes the remaining implementation-level hyperparameters and commands. Moreover, we provide more training details in Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Section 5 states that width scaling experiments are averaged over ten random seeds with standard-deviation error bars. We also report the main CIFAR-10 width scaling and depth-scaling results under this multi-seed evaluation protocol.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: Section 5.1 reports that all experiments are conducted on a single NVIDIA RTX 4090 GPU, but the current version does not provide detailed per-experiment runtime, peak memory usage, or total compute. We will include these details in the supplementary reproduction material.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

Justification: The research uses standard image classification benchmarks described in Section 5.1 and does not involve human-subject data collection, private user data, deceptive data collection, or high-risk deployment. The submission is anonymized and the experiments are reported according to the reproducibility and transparency requirements of the checklist.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [N/A]

Justification: The paper presents a foundational architecture-level analysis of logic gate networks. It does not introduce a deployed application, collect or release new data, involve human subjects, or target any sensitive decision-making setting; therefore, we do not identify direct societal impacts beyond standard research use.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release high risk models, pre-trained language models, image generators, scraped datasets, or private data. The work focuses on a logic gate-based model architecture and experiments on standard image classification benchmarks described in Section 5.1.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Section 5.1 cites all public benchmark datasets used in the experiments, and Appendix A.3 credits the original dataset creators and sources through the corresponding references. We use the standard public dataset distributions and official splits where available, follow the official licenses or terms of use where specified, and use the datasets only for benchmark evaluation. We do not redistribute, repackage, sublicense, or modify any dataset.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper introduces the proposed DLK and EDLK model components, which are described in Section 4. The experimental settings, dataset details, architecture configurations, and training protocol needed to reproduce the reported results are provided in Section 5.1 and Appendix A.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing, user studies, human-subject experiments, or newly collected human data. The experiments are conducted on standard image-classification benchmarks described in Section 5.1.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing, user studies, human-subject experiments, or newly collected human data, so IRB approval or equivalent review is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development, experiments, and analysis do not use LLMs as an important, original, or non-standard component. Any use of language tools, if any, is limited to writing, editing, or formatting and does not affect the methodology, scientific claims, or experimental results.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.