LOGIC-NATIVE TRANSFORMER: MAPPING TRANSFORMER COMPUTATIONS TO TRAINABLE LOGIC GATE NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

The growing demand for deploying Transformers on resource-constrained edge devices motivates the design of hardware-efficient neural architectures. Differentiable logic gate networks offer one such approach by directly learning compositions of the Boolean operations underlying digital hardware, enabling representation learning in feedforward, convolutional, and recurrent architectures. Extending this approach to Transformers requires computing attention with Boolean operations and overcoming the optimization difficulties of deeply stacked logic gates. To address these challenges, we propose a logic attention mechanism based on XNOR similarity, top-k selection, and majority vote, and introduce XOR residual connections. Integrating these components with learnable logic gate modules yields a scalable Logic Gate Network Vision Transformer (LGN-ViT) with a unified Boolean inference architecture. On CIFAR-10, LGN-ViT reaches a depth of 96 learnable logic gate layers, 6.4 times that of the convolutional logic gate baseline. Increasing model width improves accuracy on CIFAR-10 and MNIST as parameter counts grow. LGN-ViT also offers competitive tradeoffs between accuracy, parameter count, and estimated logic budget compared with established logic networks.

1 INTRODUCTION

Efficient inference is essential for deploying Transformers on resource-constrained edge devices (Mehta & Rastegari, 2022). Low-bit quantization reduces the precision of weights and activations to lower storage and arithmetic costs (Jacob et al., 2018; Panferov et al., 2025). However, the underlying computation remains organized around numerical operators that must be mapped to hardware, following a paradigm of fitting hardware to the algorithm (Petersen et al., 2024). Differentiable logic gate networks (DLGNs) directly learn compositions of Boolean functions through differentiable relaxation, with the selected gates and their connections defining the inference circuit after discretization (Petersen et al., 2022). Extending this logic-native paradigm to Transformers opens a route beyond arithmetic-based computation toward efficient edge inference.

DLGNs have progressed from feedforward models to convolutional and recurrent architectures (Petersen et al., 2022; 2024; Bührer et al., 2025). However, realizing a complete logic-native Transformer presents three challenges. First, attention requires input-dependent query-key similarity computation and value aggregation (Vaswani et al., 2017), which must be realized through Boolean mechanisms within a fixed circuit topology. Second, softmax produces continuous attention weights through nonlinear normalization, requiring a discrete alternative compatible with Boolean inference. Third, residual connections are essential for information propagation in deep Transformers (He et al., 2016; Vaswani et al., 2017), but conventional addition does not preserve binary states. This calls for designing Boolean residual connections that preserve and update information across layers while maintaining binary states.

To address these challenges, we propose a logic attention mechanism that uses XNOR similarity to compare queries and keys, top-k selection to choose candidate tokens, and majority vote to aggregate their value vectors. We introduce XOR residual connections that update binary states by retaining or flipping individual bits, supporting information propagation across layers. We train the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

backbone with discrete forward computation and differentiable backward surrogates to reduce the discretization gap between training and inference. Together with learnable logic gate modules, these designs form LGN-ViT, which organizes the key Transformer computations into a trainable, scalable logic gate network with a unified Boolean inference architecture. On CIFAR-10, LGN-ViT reaches 96 learnable logic gate layers, 6.4 times the depth of ConvLogic (Petersen et al., 2024). Accuracy rises with parameter count as LGN-ViT is widened, reaching 57.75% on CIFAR-10 and 98.11% on MNIST. Comparisons with established logic networks show competitive tradeoffs between accuracy, parameter count, and estimated logic budget.

2 Background and Related Work

2.1 Differentiable Logic Gate Networks

The original DLGN framework makes logic gate networks trainable through continuous relaxation (Petersen et al., 2022). Each neuron receives two inputs through fixed connections and learns a choice among the 16 Boolean functions G_i. Let $\mathbf{z} \in \mathbb{R}^{16}$ denote trainable gate-selection logits and $\widetilde{G}_i$ the continuous relaxation of G_i. For inputs $a, b \in [0, 1]$, the neuron computes

$$f_{\mathbf{z}}(a, b) = \sum_{i=1}^{16} \frac{\exp(z_i)}{\sum_{j=1}^{16} \exp(z_j)} \widetilde{G}_i(a, b). \tag{1}$$

Training optimizes the logits, while deployment selects $G_{i^\star}$ with $i^\star = \arg\max_i z_i$ at each neuron, yielding a discrete logic circuit. Building on this foundation, subsequent work has advanced DLGNs along two complementary directions by extending their architectures and improving their parameterization and training.

Along the architectural direction, convolutional DLGNs compose learnable logic gates into trees, allowing each output to combine multiple inputs (Petersen et al., 2024). A depth-two tree with three separately parameterized gates f_1, f_2, f_3 computes

$$T(a_1, a_2, a_3, a_4) = f_3(f_1(a_1, a_2),\ f_2(a_3, a_4))\,. \tag{2}$$

Applying each tree to local receptive fields with parameters shared across spatial locations yields a convolutional logic mapping. Recurrent DLGNs extend the framework to sequence modeling by updating hidden states with logic gate layers whose parameters are shared across time steps (Bührer et al., 2025).

Improvements in parameterization and training include residual initialization (RI), which favors a pass-through gate to preserve information at initialization. The original implementation sets $z_{\mathrm{id}}^{(0)} = 5$ for the pass-through gate $G_{\mathrm{id}}(a, b) = a$ and $z_i^{(0)} = 0$ for $i \neq \mathrm{id}$ (Petersen et al., 2024). For gate parameterization, Light-DLGN introduces input-wise parametrization (IWP), which directly learns the outputs for the four binary input combinations, reducing the number of parameters per two-input gate from 16 to 4 (Rüttgers et al., 2026). Let $\omega_{uv} \in [0, 1]$ denote the output associated with binary input (u, v). The relaxed gate computes

$$f_{\boldsymbol{\omega}}(a, b) = (1-a)(1-b)\omega_{00} + (1-a)b\,\omega_{01} + a(1-b)\omega_{10} + ab\,\omega_{11}. \tag{3}$$

This parametrization represents all two-input Boolean functions, and binarizing the four output values yields a discrete truth table. For training with discrete gates, Mind the Gap combines Gumbel-Softmax sampling with a straight-through estimator (Bengio et al., 2013) to reduce the discretization gap (Yousefi et al., 2025).

Convolutional and recurrent DLGNs demonstrate the architectural extensibility of logic-native computation, motivating our exploration of Transformers within this paradigm. We propose logic attention for LGN-ViT and adopt convolutional logic trees with IWP to construct its learnable backbone modules. Drawing on Mind the Gap (Yousefi et al., 2025), we train the backbone modules with discrete forward computation and differentiable backward surrogates to reduce the discretization gap.

2.2 Low-Bit Transformers

Before our exploration of DLGN-based Transformers, low-bit methods had already pursued efficient inference by reducing the numerical precision of weights and activations. BitNet uses binary

weights and 8-bit activations for language modeling (Wang et al., 2025), while QuEST enables stable training with low-bit weights and activations (Panferov et al., 2025). For vision Transformers, Bi-ViT addresses attention distortion caused by binarization (Li et al., 2024). These methods reduce the precision of numerical representations while retaining feature transformations based on numerical coefficients and weighted sums. BOLD and multi-Boolean architectures move weight optimization into the Boolean domain, eliminating the need for full-precision latent weights (Nguyen et al., 2024; Tran & Nguyen, 2026). LGN-ViT shifts the learning target from weights to Boolean gate functions for feature transformations and uses logic attention and Boolean residual updates to propagate and preserve binary token states during inference, so that logic-native computation spans feature transformations, token interactions, and cross-layer updates.

2.3 Binary Attention

Prior work on low-bit quantization has also explored binarizing attention computation. BinaryAttention quantizes queries and keys to one bit and computes their similarities using XNOR and popcount operations (Xiao et al., 2026). It adjusts the similarity scores using scaling factors and a bias term before applying softmax to obtain attention coefficients. It then quantizes these coefficients and the value vectors to 8 bits for weighted aggregation. In essence, BinaryAttention retains the numerical weighting framework of softmax attention despite using binary query and key representations.

Our logic attention also uses XNOR similarity to compare binary queries and keys. It replaces softmax normalization and numerical weighted aggregation with top-k selection of candidate tokens and majority vote over their binary value vectors. The resulting logic attention has a fully Boolean inference path.

3 Method

3.1 Overall Architecture

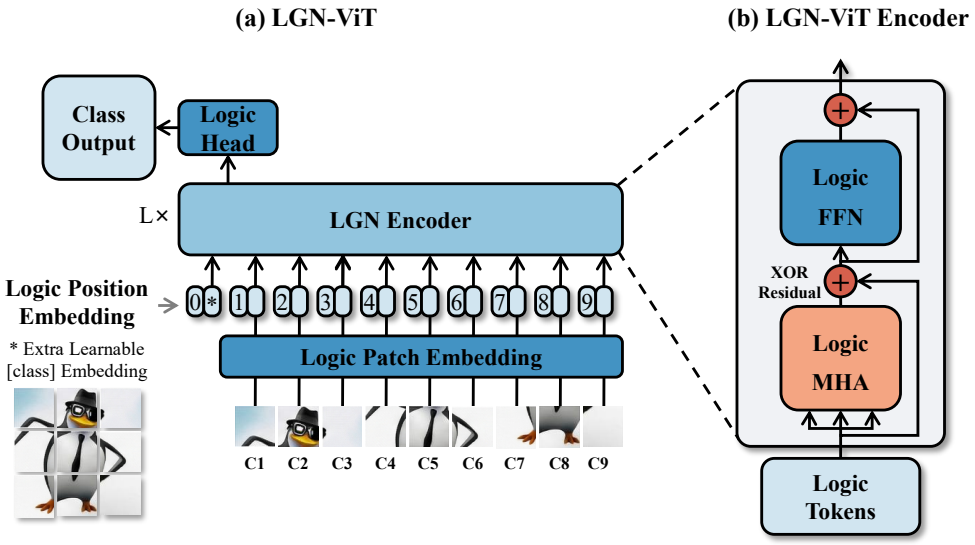

Figure 1: Overview of LGN-ViT. (a) The complete model comprises logic input embedding, L encoder blocks, and a logic classification head. The starred token denotes the learnable class token. (b) Each encoder block combines Logic MHA and Logic FFN; the circled plus signs denote XOR residual updates. Diagram layout adapted from Dosovitskiy et al. (2021).

LGN-ViT realizes a logic-native Transformer by combining two types of computational modules. Learnable logic gate modules perform input embedding, feature transformations, and classification, while fixed logic modules implement token interactions within attention and XOR residual updates.

As shown in Figure 1(a), logic input embedding converts an image into binary tokens, which pass through a stack of encoder blocks. Each block combines multi-head logic attention (Logic MHA) and a logic feed-forward network (Logic FFN) for cross-token interactions and token-wise feature transformations, respectively (Figure 1(b)). Both modules use XOR residual connections to update binary token states. After the encoder stack, a logic classification head maps the final class token to a prediction. The backbone uses discrete forward computation and differentiable backward surrogates during end-to-end training (Appendix A.4).

3.2 Learnable Logic Gate Modules

We construct the learnable logic mappings in LGN-ViT from trees of two-input gates using the IWP formulation in Section 2.1. The connections are fixed, while the gate functions are learned. Figure 2 illustrates how input organization, parameter sharing, and output grouping adapt these trees to input embedding, token-wise transformations, and classification. Appendix A.1 specifies their construction and interfaces.

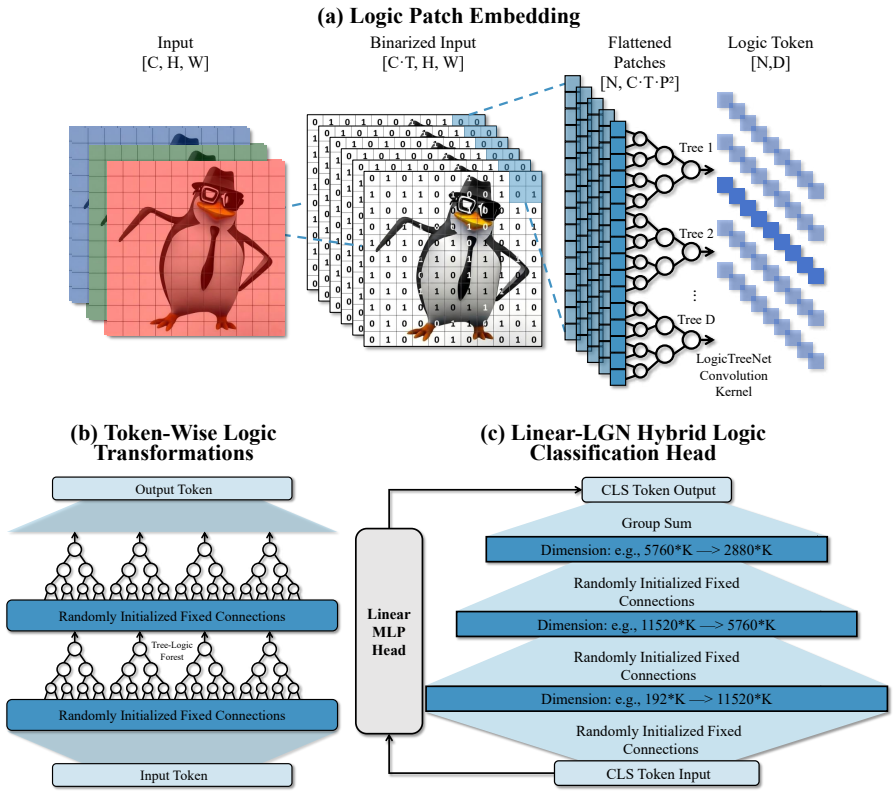

Figure 2: Learnable logic gate modules in LGN-ViT. (a) Logic input embedding uses threshold encoding and convolutional logic trees to generate image tokens. (b) Token-wise transformations for QKV, output projection, and FFN. (c) Logic classification head with a training-only auxiliary linear branch.

3.2.1 Logic Input Embedding

To provide binary inputs to the logic gate network, we encode each pixel value x using fixed thresholds as $\mathrm{Enc}_t(x) = \mathbf{1}[x > \tau_t]$, where τ_t is the t-th threshold (Petersen et al., 2024). As shown in Figure 2(a), we represent each local receptive field as a flattened vector of threshold-encoded bits across channels and spatial positions. Convolutional logic trees extract features from these local inputs, with connections and gate parameters shared across spatial positions. The input embedding outputs a sequence of N image tokens of width D. We train the trees using the continuous IWP formulation in Equation (3) and binarize their truth-table entries before deployment.

We prepend a learnable class (CLS) token for classification (Dosovitskiy et al., 2021) and encode token positions through bitwise operations that preserve binary representations. At inference, let $E \in \{0,1\}^{(N+1)\times D}$ denote the sequence before position encoding, with position $n = 0$ corresponding to CLS. For each token position n and feature bit d, a learned position parameter $p_{n,d}$ determines the operation using a fixed threshold $\epsilon_{\text{pos}} > 0$:

$$S_{0,n,d} = \begin{cases} 0, & p_{n,d} < -\epsilon_{\text{pos}}, \\ E_{n,d}, & -\epsilon_{\text{pos}} \le p_{n,d} \le \epsilon_{\text{pos}}, \\ 1, & p_{n,d} > \epsilon_{\text{pos}}. \end{cases} \tag{4}$$

This rule selects CLEAR, KEEP, or SET for each feature bit of every token, including CLS, forming the initial binary token state S_0 for the encoder stack. During training, the CLS token uses a continuous parameterization, while the position operations use discrete forward choices with differentiable backward surrogates. Before deployment, the CLS token is binarized and the position operations are fixed.

3.2.2 Token-Wise Logic Transformations

In standard Transformers, QKV projection, attention output projection, and FFN perform token-wise feature transformations (Vaswani et al., 2017). To implement these transformations in the Boolean domain, we use the tree-based logic mappings shown in Figure 2(b). Each module has its own connections and gate parameters, which are shared across token positions within that module. QKV projection maps each token to binary query, key, and value vectors, which are then split into attention heads. The output projection maps the concatenated head outputs to binary features of width D. The FFN operates on the state after the attention residual connection (Section 3.4), producing binary features of the same width. During training, these mappings use discrete truth tables for forward computation and continuous IWP surrogates based on Equation (3) to propagate gradients to their inputs and gate parameters. At inference, the learned truth tables are fixed, and the modules execute Boolean operations.

3.2.3 Logic Classification Head

The logic classification head converts the final CLS token into class scores through grouped logic outputs (Figure 2(c)). At inference, let $o_{c,g} \in \{0,1\}$ denote the g-th output bit assigned to class c, with D_{cls} output bits per class. The classification readout computes the class scores r_c and prediction $\widehat{y}$ as

$$r_c = \sum_{g=1}^{D_{\text{cls}}} o_{c,g}, \qquad \widehat{y} = \arg\max_c r_c. \tag{5}$$

GroupSum counts the active bits within each class group, and the class with the largest count is selected (Petersen et al., 2022).

During training, the logic head uses the continuous IWP formulation in Equation (3). An auxiliary linear classifier receives the same CLS token and provides an additional supervision path to the backbone. We calibrate the class scores of both branches and apply cross-entropy to their weighted combination:

$$\mathcal{L}_{\text{cls}} = \text{CE}\big(\alpha \mathbf{r}^{\text{cal}}_{\text{logic}} + (1-\alpha)\mathbf{r}^{\text{cal}}_{\text{aux}},\, y\big)\,, \tag{6}$$

where $\mathbf{r}^{\text{cal}}_{\text{logic}}$ and $\mathbf{r}^{\text{cal}}_{\text{aux}}$ denote the calibrated training scores of the logic and auxiliary branches, respectively, and y is the target label. We gradually increase $\alpha \in [0,1]$ to 1, shifting supervision to the logic branch. Before deployment, we binarize the learned truth-table entries. Inference uses only the logic head and its raw class counts; the auxiliary branch and score calibration are used only during training.

3.3 Logic Attention Core

Beyond token-wise feature transformations, attention requires content-dependent interactions between tokens. The attention core outlined in Figure 3 processes binary queries, keys, and values produced by the learnable projections. Within each head, XNOR similarity measures agreement between queries and keys, top-k selection identifies candidate tokens, and majority vote aggregates

their values into binary outputs. Together, these operations implement content-dependent token interactions through fixed Boolean rules, with no learnable parameters in the core.

The core executes discrete forward computation during both training and inference, with differentiable backward surrogates supplying gradients to the upstream projections during training. In our GPU inference implementation (Appendix A.2), we use bit packing, reductions based on lookup tables (LUTs), and a comparison tree to avoid gate-by-gate software simulation. With fixed model dimensions and bit widths, these operations can be expressed in hardware descriptions and synthesized into logic gate networks. Appendix A.3 presents a direct gate-level implementation of the same attention mechanism using XNOR matching, AND/OR bit compaction, Boolean comparisons, and multiplexing.

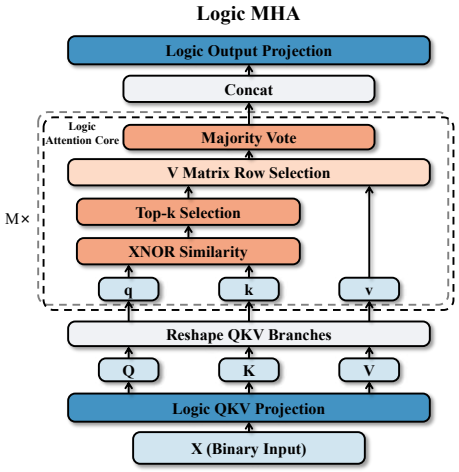

Figure 3: Logic MHA in LGN-ViT. Dark blue blocks denote learnable projections; the dashed region contains the parameter-free core replicated across M heads. X is the binary input, and q, k, and v are per-head representations. The core produces binary outputs through input-dependent token selection and aggregation.

3.3.1 XNOR Similarity

We compare binary queries and keys using the XNOR similarity shown in Figure 3 to score candidate tokens. For each of the M attention heads, let $\mathbf{q}_n, \mathbf{k}_{n'} \in \{0,1\}^{D_\mathrm{h}}$ denote the query at position n and the key at position n', where $D_\mathrm{h} = D/M$ is the head width. We define their similarity as

$$\mathrm{sim}(\mathbf{q}_n, \mathbf{k}_{n'}) = \sum_{d=1}^{D_\mathrm{h}} \mathrm{XNOR}(q_{n,d}, k_{n',d}). \tag{7}$$

The score counts both 0-0 and 1-1 matches, so larger counts indicate greater bitwise agreement. In our GPU inference implementation, we pack queries and keys into bytes, apply XNOR (Hubara et al., 2016), count matching bits within each byte using a lookup table, and combine these counts through LUT-based reduction.

Training retains the same discrete forward computation. For backpropagation, we use the continuous extension

$$\widetilde{\mathrm{sim}}(\mathbf{q}_n, \mathbf{k}_{n'}) = \sum_{d=1}^{D_\mathrm{h}} \left[q_{n,d} k_{n',d} + (1 - q_{n,d})(1 - k_{n',d}) \right]. \tag{8}$$

This expression matches the discrete score on binary inputs. Its derivatives propagate gradients from the subsequent token selection to the learnable query and key projections.

3.3.2 Top-k Selection

Using the XNOR similarity scores, we select K_sel candidate tokens per query for value aggregation, as shown in Figure 3. For $1 \le K_\mathrm{sel} < N + 1$, the selected index set for query $\mathbf{q}_n$ is

$$\mathcal{I}_n = \mathrm{TopK}_{K_\mathrm{sel}} \left([\mathrm{sim}(\mathbf{q}_n, \mathbf{k}_{n'})]_{n'=0}^{N} \right), \tag{9}$$

where TopK returns the indices of the highest scores, prioritizing larger token indices in a tie. In our GPU inference implementation, a comparison tree selects one winner at a time, excludes it from further selection, and updates the affected comparison path until K_sel candidates have been selected. The corresponding value vectors are gathered for majority vote in Section 3.3.3.

Training retains this discrete selection in the forward pass. For backpropagation, we use the continuous similarity in Equation (8) and place the selection threshold θ_n midway between the lowest selected and highest unselected scores. The continuous surrogate of the selection mask is

$$\widetilde{m}_{n,n'} = \frac{K_\mathrm{sel}\, \sigma\left(\widetilde{\mathrm{sim}}(\mathbf{q}_n, \mathbf{k}_{n'}) - \theta_n \right)}{\sum_{n''=0}^{N} \sigma\left(\widetilde{\mathrm{sim}}(\mathbf{q}_n, \mathbf{k}_{n''}) - \theta_n \right)}, \tag{10}$$

where σ denotes the sigmoid function. The derivatives of this surrogate propagate gradients from value aggregation through the continuous similarity to the learnable query and key projections.

3.3.3 MAJORITY VOTE

We aggregate the selected value vectors into a binary output through the bitwise majority vote shown in Figure 3. We choose odd K_{sel} to avoid ties and define the head output at query position n and feature bit d as

$$h_{n,d} = \mathbf{1}\left[\sum_{n' \in \mathcal{I}_n} v_{n',d} \geq \frac{K_{\text{sel}} + 1}{2}\right], \tag{11}$$

where $v_{n',d}$ is feature bit d of the value vector at position n'. The output is 1 when a strict majority of candidates have a 1 at that feature position, and 0 otherwise. In our GPU inference implementation, we pack the candidate bits for each feature position, count set bits through lookup and reduction, and apply a threshold lookup to produce the output. The head outputs are concatenated and passed to the learnable output projection.

Training retains the same discrete forward rule. Let $m_{n,n'} = \mathbf{1}[n' \in \mathcal{I}_n]$ denote the binary selection mask. For backpropagation, we use the continuous surrogate

$$\widetilde{h}_{n,d} = \sigma\left(\beta_{\text{vote}}\left[\sum_{n'=0}^{N} m_{n,n'} v_{n',d} - \frac{K_{\text{sel}} + 1}{2}\right]\right), \tag{12}$$

where $\beta_{\text{vote}} > 0$ is an inverse-temperature parameter controlling the surrogate's steepness. We evaluate its derivatives at the binary mask and value inputs. These derivatives propagate gradients to the value projection and, through the selection surrogate in Equation (10), to the query and key projections.

3.4 XOR RESIDUAL CONNECTIONS

To propagate and update binary token states across stacked encoder blocks, we place XOR residual connections around logic attention and the FFN (Figure 4). Each connection combines the module input S with a same-shaped binary update Δ, produced by the attention output projection or FFN:

$$S^{+} = S \oplus \Delta, \tag{13}$$

where $\oplus$ denotes bitwise XOR and S^{+} is the updated state. An update bit of 0 preserves the corresponding state bit, while 1 flips it. Each module thus learns how to modify the existing binary state.

We initialize the terminal gates of each update branch to produce zero-valued hard outputs, so the residual mappings begin as identities. RI controls the initial gate choices (Section 2.1), while XOR residual connections provide a persistent state path at module boundaries.

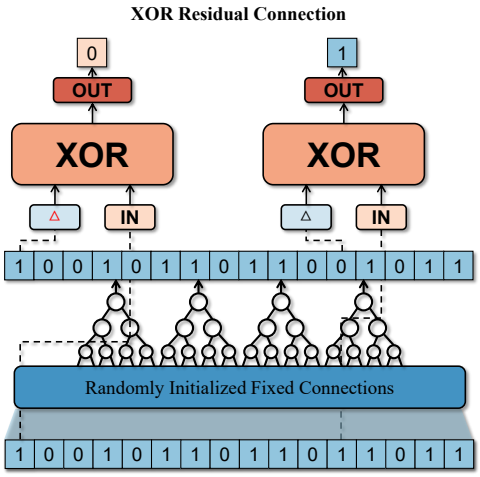

Figure 4: XOR residual connections. The logic mapping generates binary update bits, while the bypass carries the input bits to XOR. The upper examples show flipping on the left and preservation on the right.

Training and inference both execute the discrete XOR rule. For backpropagation, we evaluate the continuous XOR surrogate at the incoming state and update values. For a state bit s and an update bit δ,

$$\text{XOR}_{\text{soft}}(s, \delta) = s + \delta - 2s\delta, \qquad \frac{\partial \,\text{XOR}_{\text{soft}}}{\partial s} = 1 - 2\delta. \tag{14}$$

The direct state path has a local surrogate derivative of magnitude 1, with a positive sign for preservation and a negative sign for flipping. The same surrogate propagates gradients to the learnable update branch.

4 EXPERIMENTS

4.1 CLASSIFICATION PERFORMANCE AND MODEL SCALING

To test whether increasing model size improves classification under Boolean inference, we evaluate five LGN-ViT widths on CIFAR-10, from D192 to D5952, and two on MNIST, D192 and D384. We compare these configurations with representative logic networks (Petersen et al., 2022; 2024; Brändle et al., 2025) using test accuracy and three measures of model scale: learned scalar parameters, learnable gates, and estimated logic budgets. Learnable gates count separately parameterized two-input gate functions, with shared gates counted once across token or spatial positions. Estimated logic budgets include these gates, GroupSum, and fixed operations; Appendix B.1.4 specifies the counting rules and reuse assumptions. Appendix B.1.2 provides training configurations and MNIST readout settings; Appendix B.1.5 gives full comparisons with additional logic models. Table 1 shows that accuracy increases across all five CIFAR-10 configurations, from 45.92% to 57.75%, as the parameter count grows from 0.369M to 11.452M. MNIST exhibits the same direction of improvement, from 96.94% with 0.447M parameters to 98.11% with 0.893M parameters. These results show that larger LGN-ViT configurations can translate increased model capacity into higher classification accuracy.

Table 1: Classification accuracy and model scale on CIFAR-10 and MNIST.

Method / Configuration	Params. (M)	Learnable Gates (K)	Est. Logic Budget (K)	Test Acc. (%)
CIFAR-10				
LGN-ViT D192	0.369	88.7	188.2	45.92
LGN-ViT D384	0.658	177.4†	359.4†	48.12
LGN-ViT D576	0.947	266.1†	529.2†	50.62
LGN-ViT D768	1.478	354.8	700.4	52.56
LGN-ViT D5952	11.452	2,749.8	5,286.1	**57.75**
DiffLogicNet, 4×12K	0.768	48.0	132.0	51.27
DiffLogicNet, 4×128K	8.192	512.0	1,408.0	57.39
DiffLogicNet, 5×256K	20.480	1,280.0	3,072.0	60.78
DLGN baseline, 6×64K	6.144	384.0	832.0	50.88
ConvLogic S	1.337	83.6	160.3	60.38
ConvLogic M	10.695	668.4	1,282.6	71.01
MNIST				
LGN-ViT D192	0.447	102.1	364.0	96.94
LGN-ViT D384	0.893	204.3	677.6	**98.11**
DiffLogicNet, 6×8K	0.768	48.0	104.0	97.69
DiffLogicNet, 6×64K	6.144	384.0	832.0	98.47
ConvLogic S	1.170	73.1	145.4	98.46
ConvLogic M	4.681	292.5	581.8	99.23

Notes: † Estimated using the family-level width-scaling model. Bold marks the highest LGN-ViT accuracy on each dataset.

To place these scaling gains in context, LGN-ViT D384 achieves 98.11% on MNIST, within 0.36 percentage points of DiffLogicNet with six 64K-gate layers, while using 85.5% fewer learned parameters, 46.8% fewer learnable gates, and an 18.6% smaller estimated logic budget. It also comes within 0.35 percentage points of ConvLogic S with 23.7% fewer learned parameters. On CIFAR-10, D768 reaches 52.56%, exceeding the reported six-layer, 64K-width DLGN baseline (Brändle et al., 2025) by 1.68 percentage points while reducing learnable gates and the estimated logic budget by 7.6% and 15.8%, respectively. Together, these comparisons show that LGN-ViT benefits from model scaling and offers competitive tradeoffs between classification accuracy, learned parameter count, and estimated logic requirements.

4.2 Depth Scaling

To test whether LGN-ViT can benefit from deeper encoder stacks like conventional ViTs, we vary the number of encoder blocks from 2 to 8 on CIFAR-10 while fixing the feature width at $D = 192$ and the number of training steps (Appendix B.1.3). These configurations span 42 to 150 serial layers of learnable logic gates. With XOR residual connections, test accuracy increases overall with depth, peaking at 45.49% with 96 layers before declining gradually at greater depths (Table 2), which demonstrates effective training of LGN-ViT with 96 learnable logic gate layers, 6.4 times the 15-layer learnable gate depth of ConvLogic's CIFAR-10 architecture (Petersen et al., 2024).

Table 2: Depth scaling of LGN-ViT on CIFAR-10.

Blocks	Layers	Acc. (%)
2	42	43.98
3	60	44.35
4	78	44.33
5	96	**45.49**
6	114	44.71
7	132	44.68
8	150	44.54

4.3 Ablation Study

To evaluate the effectiveness of the proposed components, we conduct ablations on CIFAR-10 with $D = 192$, five encoder blocks, three attention heads, and a fixed training budget (Table 3; settings in Appendix B.1.3). Train and inference accuracies are measured on the same test set using their respective forward paths.

Table 3: Ablation study of LGN-ViT on CIFAR-10. Gap is Train Acc. minus Inference Acc.

Variant	Top-K Surrogate Backward	Discrete Train Forward	XOR Residual	Linear Aux. Head	Train Acc. ↑	Inference Acc. ↑	Train-Inference Gap (pp) ↓
D192 Baseline	✓	✓	✓	✓	45.21%	**45.45%**	−0.24
Top-K STE Backward	✗	✓	✓	✓	28.38%	28.52%	−0.14
No Discrete Train Forward	✓	✗	✓	✓	46.59%	37.95%	+8.64
No XOR Residual	✓	✓	✗	✓	40.87%	39.64%	+1.23
No Linear Auxiliary Head	✓	✓	✓	✗	43.20%	42.62%	+0.58

Replacing Top-K Surrogate Backward with STE reduces inference accuracy from 45.45% to 28.52%, supporting the surrogate's role in refining candidate selection by emphasizing local gradient responses near the selection boundary and attenuating them farther away. Removing Discrete Train Forward yields 46.59% train accuracy but only 37.95% inference accuracy, producing an 8.64 pp gap compared with −0.24 pp for the baseline. This discrepancy highlights the role of discrete forward training in reducing discretization error while continuous backward surrogates enable gradient-based optimization. Removing XOR residual connections lowers inference accuracy to 39.64%, supporting the role of XOR in preserving and propagating binary states across encoder blocks. Finally, removing the linear auxiliary head reduces inference accuracy to 42.62%, indicating the benefit of additional classification supervision during training. All four ablations reduce inference accuracy, which supports the effectiveness of these components in training LGN-ViT for Boolean inference. Appendices B.2 and B.3 report module-wise replacement experiments in a full-precision ViT and hardware synthesis results, respectively.

5 Conclusion

We introduced LGN-ViT, which combines learnable logic mappings, logic attention, and XOR residual connections within a unified Boolean inference architecture. Experiments demonstrate effective training at 96 learnable gate layers, accuracy gains from width expansion, and competitive tradeoffs between accuracy and model complexity. RTL validation and component synthesis further support the hardware realizability of learned feature transformations and input-dependent token interactions. Future work will focus on improving the accuracy and efficiency of logic mappings and evaluating complete hardware implementations.

AI USE STATEMENT

We used generative AI tools to assist with experimental code development and language polishing of the manuscript. The authors take full responsibility for the experimental code, reported results, and final content of this paper.

REPRODUCIBILITY STATEMENT

Appendices A and B provide method and experimental details. Code and reproduction instructions are available at `https://anonymous.4open.science/r/ViT-LGN`.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pp. 307–314. Association for Computing Machinery, 1968. doi: 10.1145/1468075.1468121. URL https://doi.org/10.1145/1468075.1468121.

Adrien Benamira, Thomas Peyrin, Trevor Yap, Tristan Guérand, and Bryan Hooi. Truth table net: Scalable, compact & verifiable neural networks with a dual convolutional small boolean circuit networks form. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 13–21, 2024. URL https://www.ijcai.org/proceedings/2024/0002.pdf.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL https://arxiv.org/abs/1308.3432.

Sven Brändle, Till Aczel, Andreas Plesner, and Roger Wattenhofer. From MNIST to ImageNet: Understanding the scalability boundaries of differentiable logic gate networks, 2025. URL https://arxiv.org/abs/2509.25933.

Simon Bührer, Andreas Plesner, Till Aczel, and Roger Wattenhofer. Recurrent deep differentiable logic gate networks. In *Proceedings of the 2nd International Workshop on Edge and Mobile Foundation Models*, pp. 31–36. Association for Computing Machinery, 2025. doi: 10.1145/3737902.3768357.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision – ECCV 2016*, volume 9908 of *Lecture Notes in Computer Science*, pp. 630–645. Springer International Publishing, 2016. doi: 10.1007/978-3-319-46493-0_38.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf.

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*, pp. 2704–2713, 2018. doi: 10.1109/CVPR.2018.00286.

Youngsung Kim. Align forward, adapt backward: Closing the discretization gap in logic gate networks, 2026. URL https://arxiv.org/abs/2603.14157.

Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1998. ISBN 9780201896855. URL https://www.informit.com/store/art-of-computer-programming-volume-3-sorting-and-searching-9780201896855.

H. W. Lang. Bitonic sorting network for n not a power of 2. Online technical note, 2023. URL https://hwlang.de/algorithmen/sortieren/bitonic/oddn.htm. Updated February 5, 2023. Accessed September 25, 2026.

Yanjing Li, Sheng Xu, Mingbao Lin, Xianbin Cao, Chuanjian Liu, Xiao Sun, and Baochang Zhang. Bi-ViT: Pushing the limit of vision transformer quantization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(4):3243–3251, 2024. doi: 10.1609/aaai.v38i4.28109.

Sachin Mehta and Mohammad Rastegari. MobileViT: Light-weight, general-purpose, and mobile-friendly vision transformer. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=vh-0sUt8HlG.

Wout Mommen, Lars Keuninckx, Matthias Hartmann, Werner Van Leekwijck, and Piet Wambacq. Fully trainable deep differentiable logic gate networks and lookup table networks, 2026. URL https://arxiv.org/abs/2607.09399.

Van Minh Nguyen, Cristian Ocampo, Aymen Askri, Louis Leconte, and Ba-Hien Tran. BOLD: Boolean logic deep learning. In *Advances in Neural Information Processing Systems*, volume 37, pp. 61912–61962. Curran Associates, Inc., 2024. doi: 10.52202/079017-1978.

Andrei Panferov, Jiale Chen, Soroush Tabesh, Mahdi Nikdan, and Dan Alistarh. QuEST: Stable training of LLMs with 1-bit weights and activations. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 47820–47836. PMLR, 2025. URL https://proceedings.mlr.press/v267/panferov25a.html.

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

Felix Petersen, Hilde Kuehne, Christian Borgelt, Julian Welzel, and Stefano Ermon. Convolutional differentiable logic gate networks. In *Advances in Neural Information Processing Systems*, volume 37, pp. 121185–121203. Curran Associates, Inc., 2024. doi: 10.52202/079017-3851.

Lukas Rüttgers, Till Aczel, Andreas Plesner, and Roger Wattenhofer. Light differentiable logic gate networks. In *International Conference on Learning Representations*, pp. 66744–66776, 2026. URL https://openreview.net/forum?id=EaGQ5luZtf.

Ba-Hien Tran and Van Minh Nguyen. Highly efficient and effective LLMs with multi-boolean architectures. In *International Conference on Learning Representations*, pp. 3976–4017, 2026. URL https://openreview.net/forum?id=r0CH5dF3Se.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Hongyu Wang, Shuming Ma, Lingxiao Ma, Lei Wang, Wenhui Wang, Li Dong, Shaohan Huang, Huaijie Wang, Jilong Xue, Ruiping Wang, Yi Wu, and Furu Wei. BitNet: 1-bit pre-training for large language models. *Journal of Machine Learning Research*, 26(125):1–29, 2025. URL https://jmlr.org/papers/v26/24-2050.html.

Shengpu Wang, Yuhao Mao, Yani Zhang, and Martin Vechev. Learning compact Boolean networks, 2026. URL https://www.mins.ee.ethz.ch/pubs/files/boolnet.pdf.

Chaodong Xiao, Zhengqiang Zhang, and Lei Zhang. BinaryAttention: One-bit QK-attention for vision and diffusion transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12106–12117, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Xiao_BinaryAttention_One-Bit_QK-Attention_for_Vision_and_Diffusion_Transformers_CVPR_2026_paper.html.

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

A Additional Method Details

A.1 Learnable Logic Gate Module Details

The token-wise mappings in Figure 2(b) stack forests of binary logic trees following convolutional DLGNs (Petersen et al., 2024). Each tree samples its input indices once with replacement to fix leaf connections. IWP gates (Equation (3)) combine adjacent pairs until one output remains; parallel tree outputs feed the next forest.

For input embedding (Figure 2(a)), thresholds are evenly spaced in $(0, 1)$, and encoded channels are ordered by image channel, then threshold. Row-major serialization determines the image-token order in E (Section 3.2.1).

The Q, K, and V branches use separate connections and gate parameters, each partitioning its output into M contiguous heads of D_{h} features. The classifier (Figure 2(c)) maps the final CLS through successive logic layers into contiguous D_{cls}-bit class groups. Table 4 lists interfaces and parameter sharing. Appendix A.4 covers training, initialization, and discretization; Appendix B.1.2 gives model configurations.

Table 4: Structural interfaces of the learnable logic gate modules at inference. The batch dimension is omitted.

Module	Input $\rightarrow$ output	Connection scope	Parameter sharing
Input embedding	Local encoded inputs $\rightarrow$ $N \times D$ tokens	Local receptive fields	Across spatial locations
QKV projections	Per-token D features $\rightarrow$ three D-dimensional vectors	Within each branch	Across token positions
Attention output projection	Concatenated MD_{h} features $\rightarrow D$ features	Within each token	Across token positions
FFN	D features $\rightarrow D$ features	Within each token	Across token positions
Classification mapping	Final D-dimensional CLS $\rightarrow$ D_{cls} bits per class	Preceding logic-layer outputs	Across input samples

A.2 GPU Implementation of the Attention Core

Within each head (Figure 3), queries and keys are packed into 8-bit words, most significant bit first, for XNOR-count matching (Hubara et al., 2016). Unused bits in the zero-padded final word are masked after XNOR. Given these words and the valid input width, $\mathrm{LUTCount}$ sums popcounts from a 256-entry table through pairwise lookup-based addition, pairing unpaired counts with zero.

Input width determines count bounds and storage; multi-byte addition propagates carries from lower to higher bytes. This yields the matching counts in Equation (7).

For each query, we use a winner tree (Knuth, 1998) whose leaves store matching counts and token indices. The leaf count is padded to the smallest power of two no smaller than $N + 1$. Padded and previously selected leaves receive the minimum representable integer score and index -1. Each internal node records the winning leaf position, prioritizing higher scores and, on a tie, larger token indices, as specified in Section 3.3.2.

Algorithm 1 GPU Inference for the Attention Core

Input (one attention head): Binary vectors $\{\mathbf{q}_n, \mathbf{k}_n, \mathbf{v}_n\}_{n=0}^{N}$, each of width D_{h}; odd K_{sel} satisfying $1 \leq K_{\mathrm{sel}} < N + 1$.
Output: Binary output vectors $\{\mathbf{h}_n\}_{n=0}^{N}$, each of width D_{h}.

Pack all query and key vectors along the feature dimension.
for $n = 0, \ldots, N$ **do**
  for $n' = 0, \ldots, N$ **do**
    Apply XNOR to the packed representations of $\mathbf{q}_n$ and $\mathbf{k}_{n'}$.
    Mask unused positions in the final word.
    $\mathrm{sim}(\mathbf{q}_n, \mathbf{k}_{n'}) \leftarrow \mathrm{LUTCount}(\text{masked XNOR words}, D_{\mathrm{h}})$
  end for
  Create leaves $(\mathrm{sim}(\mathbf{q}_n, \mathbf{k}_{n'}), n')$ for all key positions.
  Pad with invalid leaves to the next power of two.
  Propagate winning leaf positions from the leaves to the root.
  $\mathcal{I}_n \leftarrow \emptyset$
  repeat K_{sel} **times**
    $n' \leftarrow$ token index of the root winner
    $\mathcal{I}_n \leftarrow \mathcal{I}_n \cup \{n'\}$
    Assign the winning leaf the invalid score and index.
    Recompute winners along its path to the root.
  end repeat
  for $d = 1, \ldots, D_{\mathrm{h}}$ **do**
    Pack $\left(v_{n',d}\right)_{n' \in \mathcal{I}_n}$ along the candidate dimension.
    Zero-pad the final partial word.
    $h_{n,d} \leftarrow \mathrm{VoteLUT}[\mathrm{LUTCount}(\text{packed candidate bits}, K_{\mathrm{sel}})]$
  end for
end for
return the output vectors $\mathbf{h}_0, \ldots, \mathbf{h}_N$.

For each feature, selected value bits are packed along the candidate dimension, with zero padding contributing no votes. The threshold table $\mathrm{VoteLUT}$ is indexed by counts from 0 to K_{sel} and stores the strict-majority outputs defined in Equation (11).

A.3 Gate-Level Implementation of the Attention Core

For each query–key pair, we compact the matching bits in Equation (7) into a monotone sequence of zeros followed by ones, preserving their count. For binary inputs a, b, an ascending comparison cell outputs $(a \wedge b, a \vee b)$; a descending cell reverses these outputs. We organize the cells into bitonic networks (Batcher, 1968), using the arbitrary-length recursion described by Lang (2023). The first half, rounded down, is sorted in the opposite direction and the remainder in the target direction. Merging compares valid pairs separated by the largest power of two smaller than the current length, then recursively merges the leading block of that size and the remainder. Recursion stops at length one. We denote row compaction with ones on the right by $\mathcal{C}_{\rightarrow}$, and independent column compaction with ones at the top by $\mathcal{C}_{\uparrow}$.

For query position n, we stack the compacted rows in token-index order and add an all-zero column at $j = 0$ and an all-one column at $j = D_{\mathrm{h}}+1$, forming $\mathbf{U}_n \in \{0,1\}^{(N+1)\times(D_{\mathrm{h}}+2)}$. Candidate n' occupies row $n'+1$. Each entry carries its original token index encoded in $w_{\mathrm{idx}} = \lceil \log_2(N+1) \rceil$ bits. Columns are compacted upward while routing these tags. At each comparison cell, the swap signal is $\chi_{\mathrm{swap}} = a \wedge \neg b$ for ascending order and $\chi_{\mathrm{swap}} = \neg a \wedge b$ for descending order. Corresponding tag

bits t_a, t_b are routed through

$$\mathrm{Mux}(\chi_{\mathrm{swap}}; t_a, t_b) = (\neg\chi_{\mathrm{swap}} \wedge t_a) \vee (\chi_{\mathrm{swap}} \wedge t_b). \tag{15}$$

Reversing the two tag inputs gives the other output. Let $\mathbf{U}_n^{\uparrow}$ denote the resulting bit matrix, with output positions in each column numbered from 1 to $N+1$. The selection boundary is identified by

$$e_{n,j} = (\mathbf{U}_n^{\uparrow})_{K_{\mathrm{sel}},j} \wedge \neg(\mathbf{U}_n^{\uparrow})_{K_{\mathrm{sel}},j-1}, \qquad j = 1, \ldots, D_{\mathrm{h}} + 1. \tag{16}$$

The boundary columns ensure a unique transition, producing a one-hot vector $\mathbf{e}_n \in \{0,1\}^{D_{\mathrm{h}}+1}$. This vector controls multiplexers that select the boundary column and its predecessor, together with their tags.

Tags attached to ones in the preceding column identify candidates strictly above the cutoff. Tags appearing among the ones in the boundary column but not among those in its predecessor identify candidates tied at the cutoff. These memberships define the higher-score and tied-score flags for each original token index. Membership is computed by OR-reducing tag-equality matches gated by the associated bits; each equality test AND-reduces bitwise XNORs. A fixed comparison network sorts candidate records lexicographically by the higher-score flag, tied-score flag, and token index, all in descending order. Each record comparator examines the most significant differing bit of this priority key, and multiplexers route the associated indices and value vectors. The first K_{sel} records retain all strictly higher-scoring candidates and fill the remaining positions with the largest tied indices, producing $\mathcal{I}_n$ under the rule in Equation (9).

The routed values form $\mathbf{V}_n^{\mathrm{sel}} \in \{0,1\}^{K_{\mathrm{sel}} \times D_{\mathrm{h}}}$, with one row per selected candidate. Column compaction then gives the head output as

$$h_{n,d} = \left[\mathcal{C}_{\uparrow}\left(\mathbf{V}_n^{\mathrm{sel}}\right)\right]_{(K_{\mathrm{sel}}+1)/2,\, d}. \tag{17}$$

For odd K_{sel}, this position implements the strict-majority rule in Equation (11). Algorithm 2 summarizes the single-head computation. Comparison schedules and connections are fixed by the model dimensions, while input-dependent control bits determine the routing.

Algorithm 2 Gate-Level Computation of the Attention Core

Input (one attention head): Binary vectors $\{\mathbf{q}_n, \mathbf{k}_n, \mathbf{v}_n\}_{n=0}^{N}$, each of width D_{h}; odd K_{sel} satisfying $1 \le K_{\mathrm{sel}} < N+1$.
Output: Binary vectors $\{\mathbf{h}_n\}_{n=0}^{N}$, each of width D_{h}.

Encode token indices $0, \ldots, N$ as w_{idx}-bit tags.
for $n = 0, \ldots, N$ **do**
  for $n' = 0, \ldots, N$ **do**
    Generate bitwise XNOR matches between $\mathbf{q}_n$ and $\mathbf{k}_{n'}$.
    Apply $\mathcal{C}_{\rightarrow}$ and store the result in row $n'+1$.
  end for
  Add the zero and one boundary columns to form $\mathbf{U}_n$.
  for $j = 0, \ldots, D_{\mathrm{h}} + 1$ **do**
    Compact column j upward with tag routing, forming column j of $\mathbf{U}_n^{\uparrow}$.
  end for
  Compute $\mathbf{e}_n$ from adjacent entries in row K_{sel} of $\mathbf{U}_n^{\uparrow}$.
  Use $\mathbf{e}_n$ to select the boundary and preceding columns with their tags.
  Recover the higher-score and tied-score flags for each original token index.
  Sort records by descending priority of the two flags and token index, routing their values.
  Collect the first K_{sel} indices in $\mathcal{I}_n$ and their value rows in $\mathbf{V}_n^{\mathrm{sel}}$.
  Compact each column of $\mathbf{V}_n^{\mathrm{sel}}$ upward using $\mathcal{C}_{\uparrow}$.
  Read row $(K_{\mathrm{sel}}+1)/2$ of the compacted result as $\mathbf{h}_n$.
end for
return $\mathbf{h}_0, \ldots, \mathbf{h}_N$.

A.4 Training and Discretization Details

Following Light-DLGN (Rüttgers et al., 2026), we parameterize each continuous truth-table entry $\omega_{uv} \in [0,1]$ by a real-valued parameter $\Omega_{uv} \in \mathbb{R}$:

$$\omega_{uv} = \frac{1 + \sin(\Omega_{uv}/\tau_{\mathrm{gate}})}{2}, \qquad (u,v) \in \{0,1\}^2, \tag{18}$$

Table 5: Training and deployment of LGN-ViT modules.

Module	Training forward computation	Backward computation	Deployment
Input embedding and logic classifier	Continuous IWP	IWP and parameterization derivatives	Fixed binary truth tables
CLS token	Continuous sigmoid	Sigmoid derivatives	Fixed binary vector
QKV projection, output projection, and FFN	Discrete truth-table lookup	Continuous IWP surrogates	Fixed binary truth tables
Position operations	Discrete CLEAR, KEEP, or SET choices	Hard KEEP path for inputs; sigmoid surrogates for position parameters	Fixed action masks
Attention core	Discrete matching, selection, and voting	Surrogates in Equations (8), (10), and (12)	Discrete attention computation
XOR residual connections	Thresholding and XOR	Surrogate in Equation (14)	Bitwise XOR

where $\tau_{\text{gate}} > 0$ controls the parameter scale. Table 5 summarizes the training computations and deployment representations of the modules.

Discrete mappings use straight-through gradient estimation (Bengio et al., 2013). For token-wise logic mappings, the forward pass thresholds gate inputs at 0.5 and evaluates discrete truth tables, while the backward pass differentiates continuous IWP compositions, including their continuous intermediate values. The CLS token uses $\sigma(\ell_d^{\text{cls}}/\tau_{\text{cls}})$, where ℓ_d^{cls} is the learnable logit for feature d and $\tau_{\text{cls}} > 0$. Position operations retain the hard choices in Equation (4). Let $\chi_{n,d}^{\text{clear}}$ and $\chi_{n,d}^{\text{set}}$ denote the binary indicators of CLEAR and SET. Their continuous surrogates are

$$\begin{aligned} \widetilde{\chi}_{n,d}^{\text{clear}} &= \sigma\left(\frac{-p_{n,d} - \epsilon_{\text{pos}}}{\tau_{\text{pos}}}\right), \\ \widetilde{\chi}_{n,d}^{\text{set}} &= \sigma\left(\frac{p_{n,d} - \epsilon_{\text{pos}}}{\tau_{\text{pos}}}\right), \end{aligned} \tag{19}$$

with $\tau_{\text{pos}} > 0$. For continuous training inputs $E_{n,d}$, gradients to $p_{n,d}$ are obtained by differentiating $E_{n,d}(1 - \widetilde{\chi}_{n,d}^{\text{clear}}) + (1 - E_{n,d})\widetilde{\chi}_{n,d}^{\text{set}}$. Gradients to $E_{n,d}$ pass through KEEP positions and vanish at CLEAR or SET positions.

In Equation (10), we differentiate through both the selection threshold θ_n and the normalization denominator, while holding the selected index set $\mathcal{I}_n$ fixed. For XOR residual connections, the forward pass thresholds the incoming values at 0.5 before applying XOR; the backward pass evaluates the continuous surrogate in Equation (14) at the unthresholded inputs.

Input embedding and token-wise logic gates use RI to favor the pass-through truth-table pattern $(0, 0, 1, 1)$, ordered by inputs $(00, 01, 10, 11)$ (Petersen et al., 2024; Rüttgers et al., 2026). For the terminal gates of each XOR update branch, we multiply the initial continuous entries by a factor below 0.5, apply a small positive floor, and recover the real-valued parameters through the inverse sine transformation. All four entries then decode to zero while retaining variation in their continuous values. Position parameters are initialized to zero, selecting KEEP, and CLS logits are initialized from a zero-centered truncated normal distribution.

For the classification loss in Equation (6), let $\mathbf{r}_{\text{logic}}$ denote the class-wise sums of continuous logic-head outputs and $\mathbf{r}_{\text{aux}}$ the auxiliary linear scores. We calibrate these scores as

$$\begin{aligned}\mathbf{r}_{\text{logic}}^{\text{cal}} &= \frac{\text{Norm}_{\text{class}}(\mathbf{r}_{\text{logic}}/\tau_{\text{sum}})}{\tau_{\text{out}}},\\ \mathbf{r}_{\text{aux}}^{\text{cal}} &= \frac{\text{Norm}_{\text{class}}(\mathbf{r}_{\text{aux}})}{\tau_{\text{out}}}.\end{aligned} \tag{20}$$

Here, $\text{Norm}_{\text{class}}$ applies layer normalization across the classes of each example, without learnable affine parameters (Ba et al., 2016). For an input score vector $\mathbf{r}$, it subtracts the class mean and divides by $\sqrt{\text{Var}_c(\mathbf{r}) + \epsilon_{\text{norm}}}$, where $\epsilon_{\text{norm}} > 0$. The positive parameters τ_{sum} and τ_{out} set the pre-normalization and final score scales, respectively. With normalized training progress $\rho \in [0, 1]$, supervision shifts to the logic branch according to

$$\alpha(\rho) = \sin\left[\frac{\pi}{2}\,\text{clip}\left(\frac{\rho - \rho_{\text{start}}}{\rho_{\text{end}} - \rho_{\text{start}}}, 0, 1\right)\right], \tag{21}$$

where $0 \le \rho_{\text{start}} < \rho_{\text{end}} \le 1$. Thus, α increases from 0 to 1 over the specified interval.

Position regularization applies to image-token positions $n = 1, \ldots, N$, excluding CLS, and uses the hard action masks with their straight-through gradients. It combines three penalties: the squared excess of the non-KEEP fraction above a prescribed budget, spatial total variation of $\chi^{\text{set}} - \chi^{\text{clear}}$, and the squared difference between the mean SET and CLEAR fractions. Spatial total variation equally averages the mean absolute differences between horizontally and vertically adjacent positions. The weighted sum of these penalties augments $\mathcal{L}_{\text{cls}}$ to form the training objective.

For deployment, each binary truth-table entry $\widehat{\omega}_{uv}$ is set to 1 when the phase $\Omega_{uv}/\tau_{\text{gate}}$, reduced modulo 2π, lies in $[0, \pi)$, and to 0 otherwise. The same decoder supplies the truth tables used in discrete training. The CLS token is fixed to $\mathbf{1}[\ell_d^{\text{cls}} \ge 0]$, and inference follows the fixed position rule and logic-head readout in Equations (4) and (5). Appendix B.1.2 specifies temperature values, schedule boundaries, regularization settings, and position-parameter freezing schedules.

B Experimental Details and Additional Results

B.1 Experimental Settings and Full Comparison Results

B.1.1 Datasets and Evaluation Protocols

We use the official training and test sets of CIFAR-10 and MNIST. A stratified split reserves 10% of each training set for validation, yielding 45,000/5,000 training/validation images for CIFAR-10 and 54,000/6,000 for MNIST. Both test sets contain 10,000 images. Pixel values are scaled to $[0, 1]$ before threshold encoding, with no channel-wise standardization. The CIFAR-10 augmentation pipeline uses random 32×32 crops with four-pixel padding and horizontal flips; MNIST, depth-scaling, and ablation runs use unaugmented inputs. Validation and test evaluation always use unaugmented images. We select checkpoints by validation accuracy and report test accuracy after fixing the model. Published comparison results retain the evaluation protocols of their respective sources.

B.1.2 Model Configurations and Training Settings

Table 6 summarizes the architecture settings underlying the width comparisons. Each QKV branch, attention output projection, and FFN contains two consecutive depth-three logic forests. The CIFAR-10 reference recipe uses AdamW with a base learning rate of 5×10^{-4}, zero weight decay, and a total batch size of 512. The reference training budget is 175,001 updates. Larger models distribute the batch across GPUs. We fix $\tau_{\text{gate}} = \tau_{\text{cls}} = \tau_{\text{pos}} = \tau_{\text{out}} = 1$ throughout training. The top-k surrogate temperature is 1, while the majority-vote temperature decreases from 3 to 2 between 10% and 80% of training. The logic-head mixing coefficient rises from 0 to 1 between 10% and 60% of training, following Equation (21), with GroupSum scale $\tau_{\text{sum}} = 100$. Position operations use $\epsilon_{\text{pos}} = 0.15$ and a non-KEEP budget of 0.15; the budget, spatial-variation, and SET/CLEAR balance penalties have weights 1, 0.02, and 0.001, respectively. Position parameters are frozen after 60% of training.

Table 6: Architecture settings for the main LGN-ViT comparisons. Token counts include CLS.

Setting	CIFAR-10	MNIST
Feature width D	192, 384, 576, 768, 5952	192, 384
Encoder blocks L	5	3
Attention heads M	3	3
Tokens $N+1$	65	197
Selected tokens K_{sel}	15	7
Input thresholds	3	3
Spatial token grid	8×8	14×14
Classification input	Final CLS	Regional binary features

For MNIST, we first train the encoder and then freeze it to train a spatial logic readout. The selected D192 and D384 encoders are taken at 125,000 and 75,000 updates, respectively, with top-k surrogate temperatures of 1 and 0.5. Their 14×14 spatial tokens are divided into four 7×7 regions. For each region and channel, three integer quantile thresholds fitted to training-set bit counts produce $12D$ binary features using strict threshold comparisons. A new logic head maps these features through widths $180D$, $90D$, and $45D$, allocating equal tree capacity to each class before ten-class GroupSum. The readout uses AdamW, batch size 512, and zero weight decay. We train it for 20,000 updates with a cosine learning rate from 2.5×10^{-4} to 10^{-6}, then refine the validation-selected head for up to 80,000 updates from 2×10^{-5} to 5×10^{-7}. The two stages use 1,000 and 500 warmup updates, respectively. Discrete readout supervision uses a GroupSum scale of 30 and a loss weight of 4, together with continuous surrogate supervision. The final predictions use the discrete logic readout.

B.1.3 Depth Scaling and Ablation Settings

The depth sweep fixes $D = 192$, three attention heads, top-15 selection, batch size 512, and 175,001 training updates, while varying L from two to eight. The input embedding and classification head each contribute three serial gate layers; QKV, attention output projection, and FFN each contribute six per encoder block. Thus, the learnable gate depth is $3 + 18L + 3 = 18L + 6$, counting one of the parallel QKV branches and excluding fixed attention and residual operations. This gives the 42–150 layers reported in Table 2.

The ablations use the same training budget and a five-block D192 reference. Top-K STE Backward replaces the boundary-sensitive selection surrogate with an identity straight-through derivative, retaining discrete selection in the forward pass. No Discrete Train Forward optimizes the continuous training path instead of the discrete token-wise mappings. No XOR Residual replaces each XOR state update with its branch output and uses ordinary residual initialization for the logic trees. No Linear Auxiliary Head fixes the logic-head coefficient to 1 from the start, removing auxiliary-head supervision. For Table 3, both accuracies are evaluated on the same test images with the same checkpoint and the logic classification head alone. Train Acc. uses the training-side forward computations, while Inference Acc. uses the discretized model. Their signed difference, Train Acc. minus Inference Acc., is reported in percentage points.

B.1.4 Parameter and Logic Budget Accounting

Learned parameters and gates. Params. counts learned scalar parameters associated with the reported model. For CIFAR-10 LGN-ViT, this includes the training auxiliary head; for MNIST, it includes the retained encoder and new readout, excluding the discarded original heads. Learnable Gates counts separately parameterized two-input gates before constant folding, counting a shared gate once across all token or spatial positions. Two gates remain distinct even if their learned truth tables coincide. IWP assigns four scalars to each gate, whereas the softmax parameterization in DiffLogicNet and ConvLogic assigns sixteen (Petersen et al., 2022; 2024). We use the published architectures and parameterizations to compute comparison counts. For models with learned connectivity, Params. is reported only when the connection parameters can also be counted.

Estimated logic budgets. The budget adds learnable gates, a GroupSum estimate of seven two-input gate equivalents per output bit, and the fixed operations covered by the accounting model. For LGN-ViT, these include spatial reductions, XOR residuals, similarity computation, comparisons, value multiplexing, and majority voting; MNIST also includes regional counting and threshold comparisons. The attention estimate assumes three head engines with two score lanes per head, sharing selection and aggregation circuitry across candidates, queries, and encoder blocks. The budgets describe combinational logic under these reuse assumptions, excluding storage, control, clocking, and routing. Component synthesis measurements are reported separately in Appendix B.3. The † entries use family-level width extrapolation, including a proportionally scaled classification head for CIFAR-10 D384 and D576. Counts unavailable under these definitions are left unreported, including the two-input gate accounting for TTnet's multi-input truth-table filters.

B.1.5 Complete Comparison Results

Tables 7 and 8 extend Table 1 with additional configurations and logic-network families. Alongside DiffLogicNet and ConvLogic (Petersen et al., 2022; 2024), we include Gumbel LGNs (Yousefi et al., 2025), LILogicNet (Fojcik et al., 2026), CAGE (Kim, 2026), and CompactLogic (Wang et al., 2026). The MNIST comparison also includes fully trainable LGNs (Mommen et al., 2026), the DLGN configurations studied by Brändle et al. (2025), and TTnet (Benamira et al., 2024). Published ConvLogic results and our additional width configurations are listed separately. The D192 hardware checkpoint is also identified separately from the main classification result.

Table 7: Full comparison on CIFAR-10. Parameters and logic counts follow Appendix B.1.4.

Method / Configuration	Params. (M)	Learnable Gates (K)	Est. Logic Budget (K)	Test Acc. (%)
LGN-ViT D192	0.369	88.7	188.2	45.92
LGN-ViT D384	0.658	$177.4^{\dagger}$	$359.4^{\dagger}$	48.12
LGN-ViT D576	0.947	$266.1^{\dagger}$	$529.2^{\dagger}$	50.62
LGN-ViT D768	1.478	354.8	700.4	52.56
LGN-ViT D5952	11.452	2,749.8	5,286.1	**57.75**
LGN-ViT D192 (hardware)	0.369	88.7	188.2	45.74
DiffLogicNet, 4×12K	0.768	48.0	132.0	51.27
DiffLogicNet, 4×128K	8.192	512.0	1,408.0	57.39
DiffLogicNet, 5×256K	20.480	1,280.0	3,072.0	60.78
DiffLogicNet, 5×512K	40.960	2,560.0	6,144.0	61.41
DiffLogicNet, 5×1024K	81.920	5,120.0	12,288.0	62.14
DLGN baseline, 6×64K	6.144	384.0	832.0	50.88
Gumbel LGN, 12×256K	49.152	3,072.0	4,864.0	$\approx 57.5^{\ddagger}$
LILogicNet S	n/a	8.0	64.0	55.11
LILogicNet M	n/a	64.0	512.0	57.66
LILogicNet L	n/a	256.0	1,152.0	60.98
Gumbel-ST+CAGE, 6×128K	12.288	768.0	1,664.0	58.00^{*}
ConvLogic S	1.337	83.6	160.3	60.38
ConvLogic M	10.695	668.4	1,282.6	71.01
ConvLogic B	n/a	2,484.0	4,859.0	80.17
ConvLogic L	n/a	4,967.0	9,718.0	84.99
ConvLogic G	n/a	6,684.0	12,826.0	86.29
ConvLogic, custom $k = 9$	$0.376^{\dagger}$	$23.5^{\dagger}$	$45.1^{\dagger}$	48.76
ConvLogic, custom $k = 18$	$0.752^{\dagger}$	$47.0^{\dagger}$	$90.2^{\dagger}$	53.08
ConvLogic, custom $k = 27$	$1.128^{\dagger}$	$70.5^{\dagger}$	$135.3^{\dagger}$	58.36
ConvLogic, custom $k = 32$	$1.337^{\dagger}$	$83.6^{\dagger}$	$160.3^{\dagger}$	58.07
CompactLogic T	1.290	80.6	360.6	62.34
CompactLogic S	2.580	161.3	721.3	66.92
CompactLogic M	5.161	322.6	1,442.6	70.21
CompactLogic L	20.644	1,290.2	5,770.2	72.67

Notes: M and K denote 10^6 and 10^3; n/a denotes an unavailable count under our definitions. Bold marks the highest LGN-ViT accuracy. † Family-level width estimates. ‡ Approximate reading from the source curve. $*$ Minimum accuracy across the source temperature sweep. The hardware-checkpoint row is separate from the main width series.

Table 8: Full comparison on MNIST. Parameters and logic counts follow Appendix B.1.4.

Method / Configuration	Params. (M)	Learnable Gates (K)	Est. Logic Budget (K)	Test Acc. (%)
LGN-ViT D192	0.447	102.1	364.0	96.94
LGN-ViT D384	0.893	204.3	677.6	**98.11**
DiffLogicNet, 6×8K	0.768	48.0	104.0	97.69
DiffLogicNet, 6×64K	6.144	384.0	832.0	98.47
ConvLogic S	1.170	73.1	145.4	98.46
ConvLogic M	4.681	292.5	581.8	99.23
ConvLogic L	$38.191^{\dagger}$	$2,386.9^{\dagger}$	$4,720.6^{\dagger}$	99.35
CompactLogic regular-S	0.768	48.0	104.0	97.95
CompactLogic regular-M	6.144	384.0	832.0	99.10
CompactLogic conv-S	2.580	161.3	721.3	99.35
CompactLogic conv-M	5.161	322.6	1,442.6	99.41
LILogicNet, 1×4K	n/a	4.0	32.0	97.96
LILogicNet, 1×8K	n/a	8.0	64.0	98.45
LILogicNet, 2Top32-16K	n/a	32.0	144.0	98.95
Fully trainable LGN, 1×8K	n/a	8.0	64.0	98.45
Fully trainable LGN, 2×8K	n/a	16.0	72.0	98.92
DLGN baseline (Mind the Gap)	6.144	384.0	832.0	98.16
DLGN baseline ($\tau = 30$)	6.144	384.0	832.0	98.31
TTnet_{6-4} small	n/a	n/a	n/a	97.44
TTnet_{6-4} big	n/a	n/a	n/a	98.32

Notes: M and K denote 10^6 and 10^3; n/a denotes an unavailable count under our definitions. Bold marks the highest LGN-ViT accuracy. † Family-level width estimate. The two DLGN baseline rows use the results of Yousefi et al. (2025) and Brändle et al. (2025), respectively.

B.2 Module-wise Replacement in a Full-Precision ViT

To investigate accuracy bottlenecks and guide further improvements to LGN-ViT, we evaluate individual logic-module replacements in a full-precision (FP) ViT-Tiny on CIFAR-10. Each configuration replaces one module type while retaining the remaining FP architecture. Replacements within the encoder span all encoder blocks.

The reference backbone uses 4×4 patches, $L = 6$ encoder blocks, feature width $D = 192$, and $M = 3$ attention heads. Thermometer encoding and output decoding connect the Logic Attention Core to the FP projections. All models are trained from scratch, and logic computations are discrete during test evaluation.

Table 9 lists the training iterations, batch sizes, initial learning rates, and weight decay. All runs use random seed 42 and CutMix with parameter 1.0. The Mixup parameter is 0.4 for Logic Input Embedding and Logic Classification Head, and 0.5 for the other configurations.

Table 9: Training settings for the module-wise replacement experiments on CIFAR-10.

Configuration	Iterations	Batch size	Learning rate	Weight decay
FP ViT-Tiny	175,001	512	10^{-3}	0
Logic Attention Core	200,000	512	5×10^{-4}	0
Logic Input Embedding	175,000	256	1.2×10^{-3}	0.08
Logic Output Projection	175,001	512	10^{-3}	0
Logic Classification Head	100,000	512	10^{-3}	0
Logic QKV Projection	175,001	512	10^{-3}	0
Logic FFN	175,001	512	10^{-3}	0

Logic Input Embedding uses a drop-path rate of 0.05, label smoothing of 0.1, and learning-rate warmup over the first 5% of training. Logic Attention Core uses a drop-path rate of 0.1, label smoothing of 0.1, and warmup over the first 10% of training. The other configurations use zero drop-path rates and no label smoothing.

Figure 5 shows changes relative to the FP reference, while Table 10 reports test accuracy and model parameter counts.

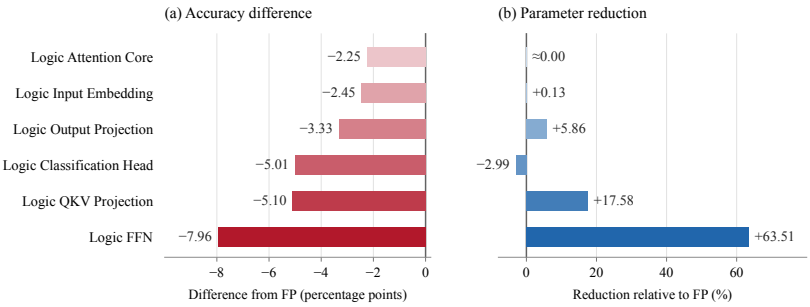

Figure 5: Module-wise replacement results on CIFAR-10 relative to FP ViT-Tiny. (a) Test accuracy differences in percentage points. (b) Relative reductions in model parameters; negative values indicate parameter increases. The FP reference achieves 92.57% accuracy with 2,693,578 parameters.

Table 10: Test accuracy and model parameter counts for individual logic-module replacements in FP ViT-Tiny on CIFAR-10.

Configuration	Test accuracy (%)	Parameters
FP ViT-Tiny	92.57	2,693,578
Logic Attention Core	90.32	2,693,596
Logic Input Embedding	90.12	2,690,122
Logic Output Projection	89.24	2,535,760
Logic Classification Head	87.56	2,774,219
Logic QKV Projection	87.47	2,220,112
Logic FFN	84.61	982,864

Note: Parameter counts refer to the complete models, including training-only components.

The retained FP modules may compensate for the representational constraints of the logic replacements during joint training, partly explaining the higher accuracy of these hybrid models relative to LGN-ViT. Among the tested configurations, Logic FFN has the largest accuracy gap from the FP reference, at 7.96 percentage points, alongside a 63.51% reduction in total model parameters. Logic QKV Projection and Logic Classification Head show gaps of 5.10 and 5.01 points, respectively. The remaining replacements have smaller gaps of 2.25–3.33 points, with Logic Attention Core closest to the reference. These results motivate future work on improving the accuracy–parameter trade-off of Logic FFN and refining Logic QKV Projection and Logic Classification Head.

B.3 Hardware Synthesis Results

To validate the hardware realizability of LGN-ViT's logic-native design, we translate its deployed Boolean computations into register-transfer level (RTL) descriptions. The learned mappings retain the exported truth tables and connections, while the remaining computations follow the prescribed attention, residual, and readout rules. We compare RTL outputs with software references using Synopsys VCS and map representative components to a 28 nm standard-cell library using Synopsys Design Compiler. The evaluated CIFAR-10 models have feature widths $D = 192$, 768, and 5952, with five encoder blocks, three attention heads, 65 tokens, and top-15 selection. All synthesis results reported below use the FF corner at 0.88 V and $0\,^{\circ}\mathrm{C}$.

The D192 reference RTL reproduces the software class counts for all four tested inputs. Tests of the D768 learned projection, classification head, similarity computation, and shared selector also pass, together with all twelve D5952 component tests. These checks establish correspondence between the RTL and software computations on the tested inputs. Table 11 reports representative D5952 standard-cell mappings, covering both a learned logic projection and fixed attention and readout operations. In particular, the combined Top-k Selection and Majority Vote circuit implements comparisons, value multiplexing, and vote accumulation within a fixed structure. Its selected tokens depend on the supplied scores, demonstrating how input-dependent token interactions can be implemented without changing circuit connectivity.

Table 11: Standard-cell synthesis of representative D5952 components.

Module	Implemented scope	Combinational cells	Sequential cells	Library area
Logic QKV Projection	V branch of the second encoder; one token	9,893	0	4,619.92
XNOR Similarity	One 1,984-bit query-key pair	6,603	0	6,569.26
Top-k Selection and Majority Vote	Three heads; 64 value bits per head	30,314	1,716	23,348.05
Top-k Selection and Majority Vote	Three heads; 256 value bits per head	71,873	4,596	55,604.68
GroupSum	Ten classes; serial accumulation	647	156	833.99

Note: Areas are synthesis-stage cell areas in library units. The selector implementations process 64 or 256 value bits per head; the full D5952 head width is 1,984 bits. Serial GroupSum accumulates 8,928 bits per class over 8,928 valid input cycles.

To characterize resource reuse within this fixed structure, we further evaluate selectors with one or three parallel head engines under matched synthesis constraints (Table 12). Each engine reuses its comparison and aggregation circuitry across the fifteen selections. For D192, three engines process all three heads in the same 45-cycle interval required by one engine to process a single head, with 2.98 times the cell area. The three-engine structure can also serve successive queries and encoder blocks. Under the same synthesis constraints, the D768 implementation supports four times the value width per head with 2.08 times the area of the D192 implementation. These results characterize concrete implementations of the fixed attention rules and their resource requirements. Together with the functional checks and learned-projection mapping, they provide hardware evidence for the Boolean feature transformations and input-dependent token interactions underlying LGN-ViT's logic-native design.

Table 12: Resource costs and processing latency of shared attention selectors.

Model	Head engines	Value bits per head	Library area	Latency (cycles)
D192	1	64	8,752.97	45
D192	3	64	26,048.86	45
D768	3	256	54,286.60	45

Note: All configurations use a 3 ns target clock, zero input/output delays, and no area limit. Latency covers selection and aggregation from externally supplied, stable scores and values, producing one head output with one engine or three head outputs with three engines. The D192 and D768 rows describe their respective RTL implementations.