FuncFormer: Circuit Representation Learning via the Flow of Functional Propagation

Yunjie Ji[1], **Zhihai Wang**[1], **Min Li**[2], **Junhua Huang**[2],
Zhihao Shi[1], **Feng Wu**[1], **Mingxuan Yuan**[2], **Jianye Hao**[3], **Jie Wang**[1*]
[1]MoE Key Laboratory of Brain-inspired Intelligent Perception and Cognition,
University of Science and Technology of China
[2]Noah's Ark Lab, Huawei [3]Tianjin University
yunjie_ji@mail.ustc.edu.cn, jiewangx@ustc.edu.cn

Abstract

Learning expressive representations for Boolean logic circuits is a fundamental challenge at the intersection of graph learning and electronic design automation (EDA). Existing circuit GNNs and graph transformers have made strong progress, but they often treat functionality as a supervision signal or auxiliary pre-training target while initializing representations from structural features. In this paper, we propose **FuncFormer**, a graph transformer that integrates classical randomized logic simulation into circuit representation learning through **functional signatures**. FuncFormer propagates sampled input patterns through gate-level logic, then uses a dual-path attention architecture to model spatial and temporal traces formed from these signatures. A simple estimator analysis shows that signature distance is an unbiased estimate of Boolean disagreement probability under the sampling distribution. Across Quality-of-Results (QoR) prediction, functional reasoning, and verification-hardness prediction, FuncFormer consistently outperforms strong circuit-learning baselines, including DeepGate4. Code is available at https://github.com/MIRALab-USTC/FuncFormer.git.

1 Introduction

Machine learning (ML) has become an increasingly important component of electronic design automation (EDA) flows Huang et al. [2021], Hamolia and Melnyk [2021], Gubbi et al. [2022], Sánchez et al. [2023], where the quality of neural circuit representations affects downstream tasks ranging from logic synthesis to formal verification. Central to this progress is the extraction of expressive embeddings from netlists, typically modeled as Directed Acyclic Graphs (DAGs) Ma et al. [2019], Kirby et al. [2019], Ghose et al. [2021], Sánchez et al. [2023], Wu et al. [2023], Wang et al. [2024a]. Although many methods leverage Graph Neural Networks (GNNs) for recursive neighborhood aggregation, standard message passing is not designed to execute Boolean semantics, so it can smooth or obscure the logic-level causal dependencies that govern circuit behavior.

Functionality-aware methods, including FGNN-style contrastive learning Wang et al. [2022, 2024b] and the DeepGate family Li et al. [2022], Shi et al. [2023, 2024], Zheng et al. [2025], have established that Boolean behavior is valuable for circuit representation learning. These methods incorporate functional information through pairwise objectives, logic-probability or truth-table supervision, and propagated signal features. FuncFormer uses sampled Boolean signatures as node inputs and organizes them into spatial and temporal trajectories for a dual-path encoder. This design targets the mismatch between topology and Boolean behavior: structurally different implementations may compute the same function, while nearly identical structures may compute different functions.

*Corresponding author.

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

In this paper, we revisit a classical EDA primitive—logic simulation—and integrate it into modern graph representation learning. We propose **FuncFormer**, a simulation-guided framework that initializes circuit nodes with functional signatures generated by randomized logic simulation. These signatures are propagated through the gate hierarchy and through latch-conditioned time frames, producing spatial and temporal traces. A dual-path transformer then encodes these traces for downstream EDA tasks. This design differs from prior functionality-aware supervision: the model is grounded in Boolean behavior at initialization time, before task-specific learning begins.

A *functional signature* is a length-d_{sim} binary vector whose k-th entry records a node's Boolean output on the k-th sampled input assignment. It is a sampled behavioral fingerprint, not a complete truth table or a certificate of equivalence.

Our analysis establishes that distances between functional signatures estimate Boolean disagreement under sampled input patterns. The result describes the information present in the input features before neural encoding. We summarize our primary contributions as follows:

• **Functional Signatures for Circuit Learning:** We integrate randomized logic simulation into circuit graph learning by initializing nodes with functional signatures. These signatures encode sampled Boolean behavior before task-specific learning and are organized into spatial and temporal traces for combinational and sequential circuits.

• **FuncFormer Architecture and Analysis:** We design a dual-path transformer that encodes spatial logic traces and temporal state traces, and show that normalized signature distance is an unbiased estimator of Boolean disagreement under the sampling distribution.

• **Empirical Evaluation in EDA Tasks:** We evaluate FuncFormer on QoR prediction, functional reasoning, and verification-hardness prediction. FuncFormer reduces average per-design QoR MSE by **26.29%** over DeepGate4, improves functional-reasoning accuracy by **2.57** percentage points, and solves **2.43**$\times$ more properties than a cone-of-influence (COI)-based schedule.

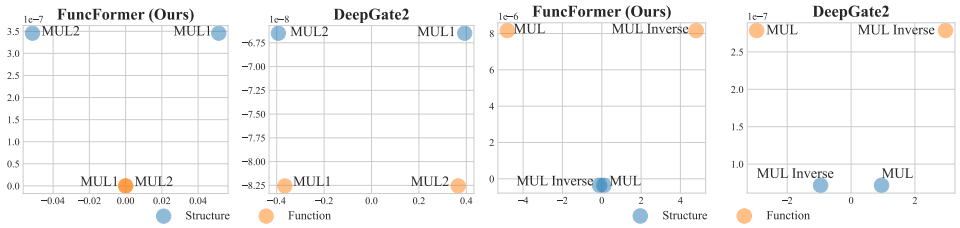
2 Motivation and preliminaries

2.1 Limitations of structural learning

Existing methodologies for circuit representation learning can be roughly divided into structural encoders and functionality-aware encoders. Standard message-passing GNNs such as GCN/GAT Kipf and Welling [2017], Veličković et al. [2018] inherit the usual limitations of local neighborhood aggregation and can miss multi-level functional dependencies. More recent circuit-specific methods, including FGNN Wang et al. [2022, 2024b] and the DeepGate family Li et al. [2022], Shi et al. [2023, 2024], Zheng et al. [2025], explicitly introduce functionality through contrastive training, logic-probability prediction, or truth-table-related supervision. These methods are important predecessors, while FuncFormer initializes nodes with sampled PI-driven signatures and organizes them into spatial and temporal trajectories. A controlled DeepGate4+PI-signature experiment in Appendix B further shows that supplying a raw signature without FuncFormer's trajectory organization does not by itself improve this benchmark.

This distinction matters because structural similarity and functional similarity are not the same property. Strong positional or random-walk encodings can break the limitations of simple message passing, but they may also make embeddings sensitive to synthesis-dependent structural changes. Conversely, local structural encoders may collapse nodes that are topologically similar but compute different Boolean functions. This creates a mismatch between graph structure and Boolean behavior: structurally similar regions may be mapped to nearby embeddings despite implementing different functions, while structurally different implementations of the same function may be mapped apart. In both cases, encoders that rely mainly on topology can produce embeddings that do not match functional equivalence or functional disagreement.

FuncFormer addresses this gap by moving functionality from the output side of learning to the input side. It initializes nodes with randomized simulation signatures and encodes spatial logic traces and temporal state traces with separate branches. The controlled functional-reasoning experiment shows that adding PI-driven signatures alone does not improve the tested DeepGate4 baseline; FuncFormer also organizes propagated signatures into depth and time trajectories.



(a) Two multipliers with different structures. (b) Multiplier and output-inverted counterpart.

Figure 1: **Functional propagation helps compare graph structure and Boolean behavior.** The points are two-dimensional PCA projections of the representations described in Appendix C; the axes are PC1 and PC2, not semantic structure/function coordinates. This qualitative illustration is followed by quantitative ablations in Section 4.3. (a) Scenario A: functionally equivalent multipliers synthesized into different structures should be close in representation space. (b) Scenario B: a multiplier and its inverted-output counterpart have opposite truth-table behavior and should be separated despite their structural similarity.

2.2 The inductive bias of functional propagation

To overcome these limitations, we use *functional propagation*—the directed propagation of signals from primary inputs (PIs) to outputs—to generate functional signatures. Logic simulation is a classical EDA primitive; FuncFormer uses it so that neural encoding starts from sampled Boolean behavior rather than from topology alone.

Ideal functional embedding vs. neural approximation. Formally, the functional identity of any node v in a combinational circuit with n inputs is uniquely determined by its truth table, which can be viewed as a vector $\mathbf{h}_v \in \{0,1\}^{2^n}$. The flow is thus the layer-wise transformation of this high-dimensional vector: $\mathbf{h}_v = f_v(\{\mathbf{h}_u \mid u \in \mathcal{P}(v)\})$. While structural embeddings rely on the *homophily assumption* (neighbors are similar), Boolean logic is inherently discrete and non-smooth (e.g., a single inverter flips the semantics entirely). We therefore use low-dimensional sampled signatures $\mathbf{x}_v \in \{0,1\}^{d_{sim}}$ ($d_{sim} \ll 2^n$) as sampled approximations of the truth table. These signatures do not replace exact Boolean equivalence, but they give the neural encoder input features whose distances reflect sampled functional disagreement.

Motivating example. The usefulness of capturing this flow is illustrated in **Figure 1**, where we compare DeepGate2 model-output embeddings with FuncFormer visualization features constructed from Boolean propagation. The detailed construction is provided in Appendix C. We use 4-bit multiplier circuits to examine two critical boundary conditions:

Scenario A (same function with different structures): Figure 1(a) depicts two multipliers implementing identical logic but synthesized into topologically distinct netlists (e.g., via different optimization recipes). A representation dominated by structural encodings may separate them even though their Boolean behavior is equivalent. In contrast, functional propagation maps them to nearby regions because their simulated input-output behavior agrees.

Scenario B (different functions with similar structures): Figure 1(b) contrasts a multiplier with its logically inverted counterpart. Despite sharing nearly identical topological skeletons (differing only by an inverter node), their truth tables disagree on every input pattern. Functional propagation directly observes this bit-vector inversion and therefore produces separated embeddings, consistent with the disagreement probability $P(y \neq \bar{y}) = 1$.

This case study illustrates that Boolean behavior provides information that graph structure alone may miss, motivating FuncFormer to include functional signatures before neural encoding.

3 The proposed approach

Problem formulation. We formulate circuit representation learning as learning an embedding function $\phi : \mathcal{V} \to \mathbb{R}^{d_{model}}$ over a heterogeneous directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Here, $\mathcal{V}$ consists of logic gates (e.g., AND, NOT) and state elements (e.g., latches), while $\mathcal{E}$ represents signal wires. Unlike standard graph representation learning where topological isomorphism often dictates similarity, our objective is to enforce **functional equivalence**: for any two nodes u, v (potentially from different

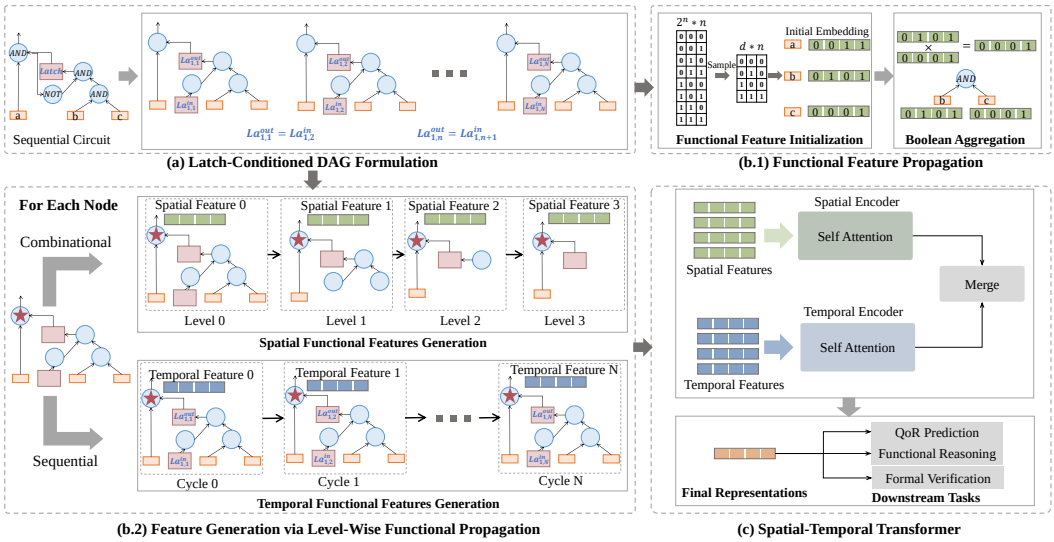


Figure 2: **Framework of FuncFormer.** The flow is (a)→(b.1)→(b.2)→(c): Boolean execution in (b.1) produces binary node signatures, (b.2) organizes those signatures into spatial and temporal sequences, and (c) encodes the sequences and combines the branch summaries. **(a) Latch-conditioned DAG formulation:** A cyclic sequential circuit is unrolled into time-conditioned DAGs by decoupling each latch into pseudo-input and pseudo-output nodes. **(b.1) Functional feature propagation:** Randomly sampled input patterns are propagated through gate-level Boolean operations to generate simulation-based node signatures. **(b.2) Level-wise feature generation:** The propagated signatures are organized into spatial trajectories across logic depth and temporal trajectories across clock cycles. **(c) Spatial-temporal transformer:** Spatial and temporal encoders process these trajectories and concatenate their embeddings before an MLP readout produces final representations for QoR prediction, functional reasoning, and verification-hardness prediction.

circuits), if they compute the same Boolean function $f_u \equiv f_v$, their embeddings should satisfy the metric constraint $\|\phi(u) - \phi(v)\| \to 0$. The core challenge is to expose Boolean behavior to the neural encoder while still handling structural variation and the feedback loops inherent in sequential designs.

Overview of FuncFormer. To address the limitations of purely structural GNNs, we introduce **FuncFormer**, a graph transformer architecture that uses functional signatures as input features. Instead of treating logic gates as generic graph nodes, FuncFormer uses the propagation traces of Boolean signals. As illustrated in Figure 2, the framework proceeds through the modules shown in the overview:

1. **Latch-conditioned DAG formulation:** To address the cyclic dependencies in sequential circuits, we formally unroll the state-transition dynamics into a sequence of time-conditioned Directed Acyclic Graphs (DAGs). This formulation (Section 3.1) allows the model to process sequential logic as a temporal trajectory, effectively decoupling instantaneous combinational logic from cross-cycle state evolution.

2. **Functional feature propagation and level-wise feature generation:** Unlike graph encoders whose input features are mainly structural or positional, such as random-walk encodings, we use randomized logic simulation to initialize node features. By propagating sampled signals through the circuit's topological hierarchy and organizing the results across logic depths and clock cycles, we generate **functional signatures** and their spatial and temporal traces before task-specific learning occurs (Section 3.2).

3. **Spatial-temporal transformer:** To process these functional signatures, we propose a dual-encoder architecture that decouples spatial reasoning from temporal aggregation. This design (Section 3.3) uses attention to capture multi-level logic dependencies within a clock cycle while modeling state transitions across time.

Collectively, these components are designed to make the learned representations reflect gate-level Boolean behavior in both combinational and sequential tasks.

3.1 Latch-conditioned DAG formulation

Sequential circuits inherently contain feedback loops introduced by state elements (latches/flip-flops), violating the acyclic assumption required for causal reasoning. To model the temporal dynamics, we employ **Time-Frame Expansion**, a standard EDA construction that unrolls the cyclic graph $\mathcal{G}$ into a sequence of N time-conditioned DAGs Kodwani et al. [2017], Roy et al. [2026], Khan et al. [2025].

Formally, let $\mathcal{S}$ be the set of state elements. We decouple the temporal evolution from instantaneous logic by splitting each latch $l \in \mathcal{S}$ into a pseudo-input l_t^{in} (current state) and a pseudo-output l_t^{out} (next state). This transformation yields a sequence of N DAGs $\{\mathcal{G}_t\}_{t=0}^{N-1}$:

$$\mathcal{G}_t = \left(\mathcal{V}_{comb} \cup \mathcal{S}_t^{in} \cup \mathcal{S}_t^{out}, \mathcal{E}_{comb}\right) \tag{1}$$

where $\mathcal{V}_{comb}$ denotes combinational gates. This formulation enables **FuncFormer** to process sequential dependencies as a trajectory of discrete state transitions, representing cyclic state feedback as a sequence of acyclic time frames. Our novelty is not time-frame expansion itself, but the use of latch-conditioned temporal traces inside a dual-path encoder. Within each time frame, the combinational And-Inverter Graph (AIG) is assumed to be a DAG, so Boolean trace propagation can be executed in topological order. FuncFormer does not introduce a separate mechanism for true combinational loops; such cases should be resolved by the EDA front-end before feature generation Brayton and Mishchenko [2010].

3.2 Functional feature propagation

A core challenge in logic representation is the exponential dimensionality of the Boolean function space. Instead of learning node embeddings from scratch, FuncFormer first generates functional signatures through logic simulation; continuous embeddings are introduced only afterward through a learnable projection layer.

Functional projection. We approximate the full truth table by sampling a subset of d_{sim} input patterns. Let $\mathbf{X} \in \{0,1\}^{d_{sim} \times |PI|}$ be a random binary matrix representing d_{sim} simulation trials for $|PI|$ primary inputs. Each column serves as the initial functional signature for a specific primary input.

Gate-level execution. To encode functional propagation across different circuit depths and improve generalization, we design a **level-wise functional propagation** mechanism. Specifically, we iteratively perform propagation by treating nodes at any j-th level ($v^j \in \mathcal{V}^j$) as pseudo-PIs and propagating their functional signatures forward.

For each starting level $0 \leq j$, the nodes at level j are re-initialized as pseudo-PIs by independently sampling fresh d_{sim}-bit functional signatures. We denote these initial signatures by $\{\mathbf{h}_{v^j}^j \mid v^j \in \mathcal{V}^j\}$, where $\mathbf{h}_{v^j}^j \in \{0,1\}^{d_{sim}}$. We then propagate these signatures along the actual fanin connections in topological order. For a node v^i at level $i > j$ with Boolean function f, its j-th functional signature is computed as:

$$\mathbf{h}_{v^i}^j = f(\{\mathbf{h}_u^j \mid u \in \mathcal{P}(v^i)\}), \tag{2}$$

where $\mathcal{P}(v^i)$ denotes the immediate fanin nodes of v^i. The level index is used to define the starting pseudo-PI set and organize the resulting trajectory; the Boolean execution itself directly indexes each gate's fanin nodes and does not require fanins to lie in the immediately preceding level. If a trace starting at level j encounters a boundary fanin from a lower level, that fanin is treated as an additional pseudo-PI for the cut induced by level j, so skip connections do not require dummy buffers. Consequently, each node v^i is represented by a spatial trace $\{\mathbf{h}_{v^i}^0, \mathbf{h}_{v^i}^1, \ldots, \mathbf{h}_{v^i}^i\}$, where each $\mathbf{h}_{v^i}^j$ corresponds to propagation starting from the j-th level. This level-wise construction captures both global and local circuit functionality through the causal execution of gate logic.

Directed Boolean aggregation. The functional propagation stage operates on binary functional signatures in $\{0,1\}^{d_{sim}}$, before any continuous neural embedding is introduced. Since the standard AIG representation contains only AND and NOT operations, the propagation rules are defined

element-wise on these signatures. For an AND node with fanin signatures $\mathbf{h}^j_{u_1}$ and $\mathbf{h}^j_{u_2}$, we compute

$$\mathbf{h}^j_v = \mathbf{h}^j_{u_1} \odot \mathbf{h}^j_{u_2}, \tag{3}$$

where $\odot$ denotes element-wise multiplication. For a NOT gate or an inverted signal, we compute the element-wise complement

$$\mathbf{h}^j_v = \mathbf{1} - \mathbf{h}^j_u. \tag{4}$$

Other gate primitives, if present in the original netlist, are first lowered to AIG form, so Boolean aggregation is fully specified by AND plus inversion. Continuous embeddings are introduced only after these signatures have been generated and organized into spatial and temporal traces.

Level-wise feature generation. Based on the level-wise propagation, we construct the final spatial and temporal traces for each node v^i:

- **Spatial Trace ($\mathbf{H}^{v^i}_{spa}$):** For the underlying combinational circuit, we aggregate the signatures generated from different starting levels into a sequence: $\mathbf{H}^{v^i}_{spa} = \{\mathbf{h}^0_{v^i}, \mathbf{h}^1_{v^i}, \ldots, \mathbf{h}^i_{v^i}\} \in \{0,1\}^{(i+1)\times d_{sim}}$. This sequence includes the node's own pseudo-PI trace, so a level-0 node has $\mathbf{H}^{v^0}_{spa} = \{\mathbf{h}^0_{v^0}\}$ rather than an empty spatial sequence. It explicitly encodes the hierarchical structural and functional information of the node's logic cone.
- **Temporal Trace ($\mathbf{H}^{v^i}_{tem}$):** For sequential logic, temporal feature generation starts from PI-driven propagation. At clock 0, specified latch initial values are fixed and unspecified latch values are sampled as pseudo-PIs. Ordinary PIs are independently re-sampled at every clock frame. For each trial, the next-state output of latch s at frame n becomes its current-state input at frame $n+1$, i.e., $La^{in}_{s,n+1} = La^{out}_{s,n}$. Temporal positions therefore denote discrete state-transition frames, not physical timestamps. The current latch inputs $\mathcal{L}_n = \{La^{in}_{s,n}\}^{|\mathcal{S}|}_{s=1}$ condition propagation in each frame, producing the node signature $\mathbf{h}^0_{v^i}(\mathcal{L}_n)$. The temporal trace has length N and is formed by the sequence: $\mathbf{H}^{v^i}_{tem} = \{\mathbf{h}^0_{v^i}(\mathcal{L}_0), \mathbf{h}^0_{v^i}(\mathcal{L}_1), \ldots, \mathbf{h}^0_{v^i}(\mathcal{L}_{N-1})\} \in \{0,1\}^{N\times d_{sim}}$.

This dual-path generation ensures that each node embedding is grounded in functional propagation across both the logic hierarchy and the temporal execution trace.

Theoretical grounding. Functional signatures have a probabilistic interpretation: distances between sampled signatures estimate distances between Boolean functions under the sampling distribution. The following result characterizes this input representation.

Theorem 3.1. *Let $x_1, \ldots, x_{d_{sim}}$ be i.i.d. uniform input assignments, and evaluate both node functions on this same set of assignments to form $\mathbf{h}_u[k] = f_u(x_k)$ and $\mathbf{h}_v[k] = f_v(x_k)$. The normalized squared Euclidean distance $\frac{1}{d_{sim}}\|\mathbf{h}_u - \mathbf{h}_v\|_2^2$ is an **unbiased estimator** of the functional disagreement probability $P(f_u(x) \neq f_v(x))$ over the uniform input distribution.*

The proof follows by observing that the squared Euclidean distance between two binary signatures is the empirical mean of d_{sim} Bernoulli disagreement indicators; Hoeffding's inequality gives concentration $2\exp(-2d_{sim}\epsilon^2)$. The use of sampled truth-table disagreement is classical and appears in functionality-aware supervision such as DeepGate2 Shi et al. [2023]; our use of the result is to characterize the information already present in FuncFormer's input signatures. Appendix D provides the full derivation and discusses non-uniform reachable-state distributions.

3.3 Spatial-temporal Transformer

The spatial trace $\mathbf{H}^v_{spa}$ and temporal trace $\mathbf{H}^v_{tem}$ encode *raw* functional information. Functional propagation and neural encoding are two separate stages: the former executes exact Boolean operations on binary signatures in $\{0,1\}^{d_{sim}}$, and the latter maps the generated traces into continuous embeddings. Each signature is transformed into the model hidden space by a learnable projection layer and then processed by Transformer-style encoders Vaswani et al. [2017], Ying et al. [2021], Chen et al. [2023]. We use separate encoders for spatial traces and temporal traces so that the model can process logic-depth information and state-transition information separately before readout.

Spatial logic reasoning. The *Spatial Encoder* processes the set of precursor states $\mathbf{H}_{spa}^{v}$. Since standard attention is permutation-invariant, we inject a logic depth embedding to preserve the topological hierarchy. The mechanism learns to weigh the contribution of different logic depths, for example by emphasizing carry-related signals in an adder.

Temporal state aggregation. Parallel to the spatial branch, the *Temporal Encoder* analyzes $\mathbf{H}_{tem}^{v}$ to capture state transitions. This is useful for verification-hardness prediction, where property difficulty often depends on evidence across multiple clock cycles.

Concatenation-MLP readout. Finally, we combine the structural and sequential insights by concatenating the spatial and temporal embeddings and applying an MLP:

$$\mathbf{z}_{final} = \mathrm{MLP}\left([\mathbf{z}_{spa} \parallel \mathbf{z}_{tem}]\right), \tag{5}$$

where $\parallel$ denotes concatenation. This simple readout keeps the branch interaction explicit: the spatial and temporal encoders first produce separate summaries, and the downstream MLP learns from their concatenated representation. We assess the contribution of each branch using single-branch ablations rather than interpreting a gating module.

4 Experiments

We evaluate *FuncFormer* on QoR estimation, functional reasoning, and verification-hardness prediction Dureja et al. [2019], Wu et al. [2023], Das et al. [2024]. These tasks jointly test synthesis-aware prediction, functional equivalence, and temporal reasoning in sequential designs. We organize the evaluation around three questions: whether functional signatures improve generalization to unseen circuit IPs, whether they help identify equivalent logic beyond graph topology, and whether temporal traces improve solver scheduling under a fixed budget.

4.1 Experimental setup

Datasets and benchmarks. For **QoR prediction**, we use OpenABC-D Chowdhury et al. [2021], which contains 870,000 AIGs from 29 design IPs; following Deng et al. [2024], 20 designs are used for training and 9 unseen IPs for testing. For **functional reasoning**, we follow the DeepGate2 setting and use 10,000 subgraphs from ITC'99 Davidson [1999], IWLS'05 Albrecht [2005], and OpenCores OpenCores [2026], testing on out-of-distribution datapath circuits. For **verification-hardness prediction**, we use HWMCC-13 Hardware Model Checking Competition [2013]; predicted hardness scores are used only to rank properties, while final outcomes are produced by the exact solver. Dataset statistics are in Appendix A.

Baselines and protocol. We compare against GCN, GAT, and DeepGate4, using the same downstream heads as prior work Chowdhury et al. [2021], Zheng et al. [2025] so that the comparison isolates the representation layer. DeepGate4 is the primary numerical DeepGate baseline; DeepGate2 is used only for the functional-reasoning protocol and the motivating visualization. Detailed hyperparameters, losses, metric aggregation, and scheduling protocol are provided in Appendix E.

Scale and generalization. The benchmark suite also evaluates transfer across circuit scales. Functional-reasoning subgraphs range from small local cones to 42k-node designs, OpenABC-D includes IPs up to 526k nodes, and HWMCC circuits contain large sequential state spaces. Thus, the evaluation is not only a within-dataset comparison: held-out OpenABC-D IPs test design-level transfer, out-of-distribution datapath circuits test functional reasoning after optimization, and HWMCC tests whether temporal traces remain useful when solver behavior depends on state evolution.

4.2 Main results

Table 1 summarizes the task-level outcomes and the capability each benchmark stresses; Tables 2 and 3 report per-circuit results for QoR prediction and functional reasoning, respectively. Full verification tables and additional results are in Appendix A. We discuss each task separately: QoR prediction tests synthesis-aware transfer, functional reasoning tests equivalence after optimization, and formal verification tests solver prioritization under a fixed budget.

Table 1: Summary of main experimental results across three EDA tasks.

Task	Evaluation focus	Benchmark	DeepGate4	FuncFormer	Gain
QoR prediction	Post-synthesis outcome from synthesis-sensitive Boolean structure	OpenABC-D	Rel. MSE: 1.00	**0.74**	-26.29%
Functional reasoning	LEC-style equivalent-gate identification after optimization	Industry Circuits	Acc.: 90.74%	**93.31%**	+2.57 pp
Hardness prediction	Solver-time class prediction for sequential properties	HWMCC-13	Acc.: 74.53%	**81.09%**	+6.55 pp
Property scheduling	Ranking exact-solver calls under a fixed verification budget	HWMCC-13	DG4/COI: 431/217	**528**	1.23/2.43×

Note: COI denotes cone-of-influence. For QoR prediction, relative MSE is averaged over held-out designs after normalizing each design by DeepGate4. Full verification and ablation details are in Appendix A; metric definitions and scheduling details are in Appendix E.

Evaluation on QoR prediction. QoR prediction estimates post-synthesis outcomes from pre-synthesis AIGs, a setting where the same high-level computation may be rewritten into substantially different gate structures. The task therefore tests whether a representation preserves Boolean organization that affects mapped area while still transferring to unseen IP blocks. FuncFormer reduces average MSE by 26.29% over DeepGate4 and improves on every held-out OpenABC-D design. The gains are especially visible on logic-intensive circuits such as `aes_xcrypt`, where long Boolean dependencies make local structural cues less reliable. At the same time, improvements on smaller SoC-style designs such as `tinyRocket` and `picosoc` show that the benefit is not limited to large netlists.

Table 2: QoR prediction on held-out OpenABC-D designs.

Circuit	GCN MSE	GAT		DeepGate4		FuncFormer (Ours)	
		MSE	Impr.	MSE	Impr.	MSE	Impr.
ethernet	1.25	1.13	9.60	0.85	32.00	**0.60**	**52.00**
wb_conmax	1.44	1.81	-25.69	1.25	13.19	**0.91**	**36.81**
jpeg	46.60	1.16	97.55	0.94	97.99	**0.67**	**98.57**
bp_be	1.72	1.07	37.79	1.12	34.88	**0.72**	**58.14**
vga_lcd	1.90	1.07	43.68	1.20	36.84	**0.73**	**61.58**
tinyRocket	0.82	0.79	3.66	0.61	25.61	**0.50**	**39.02**
picosoc	0.93	0.90	3.23	0.74	20.43	**0.67**	**27.96**
aes_xcrypt	2.34	1.73	26.07	1.55	33.76	**1.05**	**55.13**
aes_secworks	11.38	0.98	91.42	0.78	93.17	**0.65**	**94.31**
Avg. Impr.	N/A	31.91		43.09		**58.16**	

Note: MSE is lower-is-better. "Impr." is the relative MSE reduction against GCN, $(\mathrm{GCN} - \mathrm{Method})/\mathrm{GCN} \times 100$.

The per-design results in Table 2 also show a useful pattern in how the improvement appears. DeepGate4 already reduces the large errors of simple GNN baselines on many OpenABC-D designs, but FuncFormer further lowers the error after that stronger representation baseline. This is important for QoR prediction because synthesis transforms can preserve coarse graph statistics while changing the Boolean organization that determines the final mapped circuit. Functional signatures give the model direct evidence of that organization before the downstream regressor is trained, which explains why the improvement is visible across both datapath-heavy and control-oriented designs.

Evaluation on functional reasoning. Functional reasoning follows the spirit of logical equivalence checking: the model must identify gates that implement the same Boolean function after synthesis and optimization have changed the surrounding topology. Equivalent gates can be structurally distant, while structurally similar gates can differ on a small but important subset of input patterns. FuncFormer improves average accuracy from 90.74% to 93.31% over DeepGate4, with gains across all 20 out-of-distribution datapath circuits. Larger improvements tend to appear on the more difficult circuits in Table 3, suggesting that functional signatures are most helpful when topology alone gives an ambiguous signal.

Table 3: Functional reasoning accuracy on 20 unseen industrial circuits.

Circuit	Node	Edge	DeepGate4	FuncFormer (Ours)	
			Acc. (%) ↑	Acc. (%) ↑	Impr. (%) ↑
D1	39,901	60,462	94.25	**96.24**	**2.11**
D2	10,571	15,805	93.11	**95.16**	**2.20**
D3	14,059	21,067	92.92	**95.03**	**2.27**
D4	11,356	16,987	92.91	**95.08**	**2.33**
D5	39,148	59,319	83.44	**85.66**	**2.66**
D6	13,127	19,660	92.46	**94.76**	**2.48**
D7	38,340	58,096	82.62	**84.96**	**2.83**
D8	9,660	14,449	92.38	**94.79**	**2.61**
D9	14,079	21,099	92.36	**94.83**	**2.68**
D10	12,068	18,063	92.21	**94.74**	**2.75**
D11	13,968	20,929	92.08	**94.68**	**2.82**
D12	13,490	20,204	91.83	**94.61**	**3.03**
D13	10,731	16,063	91.74	**94.56**	**3.07**
D14	42,038	63,710	86.50	**89.28**	**3.20**
D15	12,643	18,927	91.51	**94.35**	**3.10**
D16	13,782	20,649	91.47	**94.37**	**3.17**
D17	10,244	15,329	91.28	**94.24**	**3.24**
D18	31,399	47,564	87.56	**90.58**	**3.45**
D19	9,659	14,425	91.10	**94.18**	**3.38**
D20	8,619	12,858	91.04	**94.19**	**3.45**
Avg.	**18,444**	**27,783**	90.74	**93.31**	**2.84**

Note: "Impr." is the per-circuit relative accuracy improvement over DeepGate4, $(\mathrm{Ours} - \mathrm{DeepGate4})/\mathrm{DeepGate4} \times 100$. Bold values indicate the best result in each row.

This behavior matches the motivation in Figure 1. In an optimized datapath, semantically equivalent sub-expressions may be implemented through different local rewrites, sharing little immediate neighborhood structure. Conversely, gates inside repeated arithmetic blocks can look structurally similar while computing different intermediate functions. The functional-reasoning benchmark stresses both cases at node level. By initializing nodes with functional signatures, FuncFormer gives the contrastive objective a geometry that is already closer to functional equivalence, rather than requiring the training loss to recover this geometry only from structural inputs.

Evaluation on formal property verification. Sequential verification is computationally intensive because solver effort depends on the evolution of state across clock cycles. We evaluate this setting in two ways. First, hardness prediction measures whether the representation separates properties by solver-time class. FuncFormer reaches 81.09% average accuracy, improving over DeepGate4 by 6.55 percentage points. Second, property scheduling measures the workflow impact of the predicted hardness scores: properties are ranked and then passed to the exact solver under the same budget. This ranking lets the solver complete 528 properties in total, compared with 431 for DeepGate4 and 217 for the COI heuristic. The largest scheduling gains appear on `6s380` and `6s413`, where static cone-size heuristics provide little separation but temporal traces expose useful difficulty structure.

The verification-hardness and property-scheduling tables in Appendix A (Tables 6 and 7) provide complementary evidence. Hardness accuracy evaluates whether the representation separates easy and hard properties before scheduling, while solved count evaluates the usefulness of the induced ordering inside the verification workflow. On `6s413`, for example, FuncFormer improves hardness accuracy over DeepGate4 and increases solved properties from 47 to 79. On `6s380`, it raises the solved count from 183 to 233. These cases are informative because the gain is realized after the learned ranking is coupled to the same exact solver, showing that the temporal branch improves the ordering of solver calls rather than merely changing a standalone classification metric.

Cross-task pattern. The three results stress complementary limits of structural encoders. QoR prediction rewards synthesis-aware Boolean organization, functional reasoning rewards invariance to optimization-induced structural change, and verification scheduling rewards temporal state information. The consistent gains across these regimes indicate that functional signatures are useful as input features, rather than only as auxiliary supervision. This also clarifies why the same representation can support both combinational and sequential tasks: spatial traces capture value flow through logic depth, while temporal traces capture state-conditioned behavior.

4.3 Ablation analysis

Table 4 summarizes the ablation evidence on sequential verification-hardness prediction by grouping variants according to the design factor they test. The grouped rows isolate branch design, input initialization, simulation width, and backbone choice under the same verification-hardness metric. We report average accuracy and the change relative to the full FuncFormer configuration; full per-circuit verification and ablation results are provided in Appendix A.

Table 4: Ablation summary for sequential verification-hardness prediction. Entries are average accuracy (%) and change relative to the full FuncFormer configuration.

Factor	Variant	Acc.	Δ
Full model	FuncFormer, $d_{sim} = 256$	**81.09**	–
Branch	Spatial-only / Temporal-only	72.77 / 73.00	-8.32 / -8.09
Initialization	Random normal / PIE	76.62 / 78.36	-4.47 / -2.73
Width	$d_{sim} = 64$ / 128 / 512	72.53 / 78.15 / 81.64	-8.56 / -2.94 / +0.55
Backbone	Level-GNN + func. sig.	64.14	-16.95

Note: Δ is computed relative to FuncFormer with $d_{sim} = 256$.

Spatial and temporal branches. Spatial-only and temporal-only variants reach 72.77% and 73.00%, respectively, while the full model reaches 81.09%. This gap suggests that verification-hardness prediction depends on both logic-cone behavior within a clock frame and latch-conditioned state evolution across frames. The two single-branch variants fail in complementary ways: the spatial branch has access to local Boolean organization but not state history, while the temporal branch observes trace evolution with weaker access to the logic structure that generates each transition. FuncFormer combines the two views so attention can select relevant logic-depth and clock-frame embeddings before readout.

Simulation width and input initialization. Very narrow signatures, such as $d_{sim} = 64$, drop to 72.53%, while increasing d_{sim} from 128 to 256 substantially improves accuracy and 512 gives only a modest additional gain. This trend matches the estimator view: wider signatures reduce sampling collisions until the sampled disagreement signal stabilizes. With the same dual-path backbone, random initialization and PIE achieve 76.62% and 78.36%, while functional signatures reach 81.09%, showing that functional signatures provide a stronger starting point on average. The comparison is especially useful because the architecture is unchanged: the gain comes from replacing topology-oriented or random inputs with signatures that already encode sampled Boolean behavior.

Backbone attribution. Functional signatures remain useful with a simpler Level-GNN backbone, which reaches 64.14% accuracy. This gap indicates that the generated traces are not merely scalar node features; they form short trajectories over logic depth and clock frames, so the encoder must select among multiple functional observations before readout. Overall, the ablations support a two-part conclusion: functional signatures provide a meaningful input space, and the dual-path transformer combines the resulting traces for downstream EDA tasks.

Across five signature-sampling runs, functional-reasoning accuracy averages 93.29% (sample SD: 0.12 percentage points); full results are in Appendix B.

5 Conclusion and limitations

FuncFormer uses sampled signatures and a fixed-budget scheduling policy; its effectiveness depends on the simulation distribution, circuit set, and temporal horizon. Retaining spatial and temporal traces also increases memory and inference costs.

FuncFormer integrates randomized logic simulation into circuit graph representation learning by using functional signatures as input features and encoding their spatial and temporal traces with a dual-path transformer. The estimator analysis characterizes signature distance as an unbiased estimator of Boolean disagreement under the sampling distribution, and the experiments show consistent gains across all three benchmarks. In formal-verification workflows, FuncFormer provides learned ranking signals for property scheduling, while exact solvers retain correctness certification. Future work can study adaptive simulation and closer integration with SAT/SMT/PDR solvers [Eén et al., 2011, Das et al., 2024, Roy et al., 2026, Dureja et al., 2019].

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant No. U23A20388). We would like to thank all the anonymous reviewers for their insightful comments. Part of this work was conducted during Yunjie Ji's internship at Huawei.

References

Christoph Albrecht. IWLS 2005 benchmarks. In *International Workshop for Logic Synthesis (IWLS)*, volume 9, 2005.

Robert Brayton and Alan Mishchenko. ABC: An academic industrial-strength verification tool. In *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*, pages 24–40. Springer, 2010.

Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. NAGphormer: A tokenized graph transformer for node classification in large graphs. In *The Eleventh International Conference on Learning Representations*, 2023.

Animesh Basak Chowdhury, Benjamin Tan, Ramesh Karri, and Siddharth Garg. OpenABC-D: A large-scale dataset for machine learning guided integrated circuit synthesis. *arXiv preprint arXiv:2110.11292*, 2021.

Sourav Das, Aritra Hazra, Pallab Dasgupta, Sudipta Kundu, and Himanshu Jain. Purse: Property ordering using runtime statistics for efficient multi-property verification. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2024.

Scott Davidson. Characteristics of the ITC'99 benchmark circuits. In *IEEE International Test Synthesis Workshop (ITSW)*, page 87, 1999.

Chenhui Deng, Zichao Yue, Cunxi Yu, Gokce Sarar, Ryan Carey, Rajeev Jain, and Zhiru Zhang. Less is more: Hop-wise graph attention for scalable and generalizable learning on circuits. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

Rohit Dureja, Jason Baumgartner, Alexander Ivrii, Robert Kanzelman, and Kristin Y Rozier. Boosting verification scalability via structural grouping and semantic partitioning of properties. In *2019 Formal Methods in Computer Aided Design (FMCAD)*, pages 1–9. IEEE, 2019.

Niklas Eén, Alan Mishchenko, and Robert Brayton. Efficient implementation of property directed reachability. In *2011 Formal Methods in Computer-Aided Design (FMCAD)*, pages 125–134. IEEE, 2011.

Amur Ghose, Vincent Zhang, Yingxue Zhang, Dong Li, Wulong Liu, and Mark Coates. Generalizable cross-graph embedding for GNN-based congestion prediction. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

Kevin Immanuel Gubbi, Sayed Aresh Beheshti-Shirazi, Tyler Sheaves, Soheil Salehi, Sai Manoj PD, Setareh Rafatirad, Avesta Sasan, and Houman Homayoun. Survey of machine learning for electronic design automation. In *Proceedings of the Great Lakes Symposium on VLSI 2022*, pages 513–518, 2022.

Vladyslav Hamolia and Viktor Melnyk. A survey of machine learning methods and applications in electronic design automation. In *2021 11th International conference on advanced computer information technologies (ACIT)*, pages 757–760. IEEE, 2021.

Hardware Model Checking Competition. Hardware model checking competition 2013. `https://fmv.jku.at/hwmcc13/`, 2013.

Guyue Huang, Jingbo Hu, Yifan He, Jialong Liu, Mingyuan Ma, Zhaoyang Shen, Juejian Wu, Yuanfan Xu, Hengrui Zhang, Kai Zhong, et al. Machine learning for electronic design automation: A survey. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 26(5):1–46, 2021.

Sadaf Khan, Zhengyuan Shi, Ziyang Zheng, Min Li, and Qiang Xu. DeepSeq2: Enhanced sequential circuit learning with disentangled representations. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, pages 498–504, 2025.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Robert Kirby, Saad Godil, Rajarshi Roy, and Bryan Catanzaro. Congestionnet: Routing congestion prediction using deep graph neural networks. In *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 217–222. IEEE, 2019.

Urvashi Kodwani, Sonal Rajurkar, and SG Mundada. Realization of sequential circuit using finite state machine. In *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 472–476. IEEE, 2017.

Min Li, Sadaf Khan, Zhengyuan Shi, Naixing Wang, Huang Yu, and Qiang Xu. DeepGate: Learning neural representations of logic gates. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 667–672, 2022.

Yuzhe Ma, Haoxing Ren, Brucek Khailany, Harbinder Sikka, Lijuan Luo, Karthikeyan Natarajan, and Bei Yu. High performance graph convolutional networks with applications in testability analysis. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

OpenCores. OpenCores: Open source hardware rtl and ip core repository. `https://opencores.org/`, 2026. Accessed: 2026-05-06.

Soumik Guha Roy, Sumana Ghosh, Ansuman Banerjee, Raj Kumar Gajavelly, and Sudhakar Surendran. MPBMC: Multi-Property Bounded Model Checking with GNN-guided Clustering, 2026.

Daniela Sánchez, Lorenzo Servadei, Gamze Naz Kiprit, Robert Wille, and Wolfgang Ecker. A comprehensive survey on electronic design automation and graph neural networks: Theory and applications. *ACM Transactions on Design Automation of Electronic Systems*, 28(2):1–27, 2023.

Zhengyuan Shi, Hongyang Pan, Sadaf Khan, Min Li, Yi Liu, Junhua Huang, Hui-Ling Zhen, Mingxuan Yuan, Zhufei Chu, and Qiang Xu. DeepGate2: Functionality-aware circuit representation learning. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.

Zhengyuan Shi, Ziyang Zheng, Sadaf Khan, Jianyuan Zhong, Min Li, and Qiang Xu. DeepGate3: Towards scalable circuit representation learning. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2024.

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*, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Zhihai Wang, Lei Chen, Jie Wang, Yinqi Bai, Xing Li, Xijun Li, Mingxuan Yuan, Jianye Hao, Yongdong Zhang, and Feng Wu. A circuit domain generalization framework for efficient logic synthesis in chip design. In *Forty-first International Conference on Machine Learning*, volume 235, pages 50163–50207. PMLR, 2024a.

Ziyi Wang, Chen Bai, Zhuolun He, Guangliang Zhang, Qiang Xu, Tsung-Yi Ho, Bei Yu, and Yu Huang. Functionality matters in netlist representation learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 61–66, 2022.

Ziyi Wang, Chen Bai, Zhuolun He, Guangliang Zhang, Qiang Xu, Tsung-Yi Ho, Yu Huang, and Bei Yu. FGNN2: A powerful pre-training framework for learning the logic functionality of circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024b.

Nan Wu, Yingjie Li, Cong Hao, Steve Dai, Cunxi Yu, and Yuan Xie. Gamora: Graph learning based symbolic reasoning for large-scale Boolean networks. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In *Advances in Neural Information Processing Systems*, volume 34, pages 28877–28888, 2021.

Ziyang Zheng, Shan Huang, Jianyuan Zhong, Zhengyuan Shi, Guohao Dai, Ningyi Xu, and Qiang Xu. DeepGate4: Efficient and effective representation learning for circuit design at scale. *arXiv preprint arXiv:2502.01681*, 2025.

A Full experimental results

This appendix provides the full per-circuit verification-hardness and property-scheduling results, additional ablation tables, and dataset statistics. Per-circuit QoR and functional-reasoning results are reported in Tables 2 and 3 in the main text.

Table 5: Benchmark statistics for the appendix experiments.

Task	**Source**	**# Nodes**	**# Edges**	**# Levels**
Functional reasoning	DeepGate2 setting	47–42k	56–64k	5–262
QoR prediction	OpenABC-D	866–526k	1.1k–733k	13–1,522
Formal verification	HWMCC	60k–270k	85k–395k	38–262

Note: Ranges are Min–Max across circuits. The functional-reasoning benchmark follows the DeepGate2 setting and uses circuits from ITC'99, IWLS'05, and OpenCores.

Table 6: Verification-hardness classification accuracy on HWMCC-13 circuits.

Circuit	**GCN Acc.**	**GAT**		**DeepGate4**		**FuncFormer (Ours)**	
		Acc.	**Impr.**	**Acc.**	**Impr.**	**Acc.**	**Impr.**
6s410	76.32	76.32	0.00	84.67	10.9	**92.89**	**21.7**
6s407	62.96	63.89	1.44	67.42	7.07	**70.37**	**11.7**
6s264	59.79	39.17	-34.5	64.15	7.3	**69.59**	**16.4**
6s380	43.98	43.98	0.00	87.32	98.6	**93.02**	**111.5**
6s413	42.05	42.05	0.00	69.11	64.4	**79.56**	**89.3**
Avg. Impr.	N/A	-6.61		37.65		**50.12**	

Note: Entries are percentages. "Impr." is the per-circuit relative accuracy change with respect to GCN, $(\text{Method} - \text{GCN})/\text{GCN} \times 100$.

Table 7: Verification-hardness-guided property scheduling.

Benchmark		**Heuristics**		**Learning-based**				
Circuit	**#Prop.**	**COI**	**Random**	**GCN**	**GAT**	**DeepGate4**	**Ours**	**Impr.**
6s410	114	47	17	68	55	70	**77**	**1.10**
6s407	108	55	27	68	46	65	**69**	**1.06**
6s264	194	32	18	64	34	66	**70**	**1.06**
6s380	266	78	55	118	223	183	**233**	**1.27**
6s413	176	5	17	9	9	47	**79**	**1.68**
Total	**858**	217	134	327	367	431	**528**	**1.23**
Avg.	**171.6**	43.4	26.8	65.4	73.4	86.2	**105.6**	**1.23**

Note: Values are solved-property counts under the same exact solver budget. "Impr." is the solved-count ratio relative to DeepGate4.

Table 8: Spatial-temporal branch ablation.

Circuit	**Spatial-only**	**Temporal-only**	**FuncFormer**
6s410	89.12	87.45	**92.89**
6s407	64.55	65.60	**70.37**
6s264	61.20	62.12	**69.59**
6s380	82.15	81.38	**93.02**
6s413	66.82	68.44	**79.56**
Average	72.77	73.00	**81.09**

Note: Entries are verification-hardness prediction accuracy (%).

Table 9: Effect of simulation width on hardness prediction.

Circuit	$d_{sim} = 64$	$d_{sim} = 128$	$d_{sim} = 256$	$d_{sim} = 512$
6s410	83.12	89.55	92.89	**93.45**
6s407	62.90	67.84	70.37	**70.92**
6s264	62.24	67.08	69.59	**70.15**
6s380	83.24	89.67	93.02	**93.58**
6s413	71.16	76.62	79.56	**80.11**
Average	72.53	78.15	81.09	**81.64**

Note: Entries are accuracy (%); gains diminish beyond $d_{sim} = 256$.

Table 10: Input-initialization ablation with the dual-path backbone.

Circuit	Random Normal	PIE	Functional Sig. (Ours)
6s410	86.4	87.8	**92.89**
6s407	70.7	**72.3**	70.37
6s264	71.9	**74.1**	69.59
6s380	88.9	90.5	**93.02**
6s413	65.2	67.1	**79.56**
Average	76.62	78.36	**81.09**

Note: Entries are accuracy (%). Random Normal and PIE remove simulation-guided functional propagation from the input features; Functional Sig. uses the proposed functional signatures.

Table 11: Backbone attribution with functional signatures.

Circuit	Level-GNN + Func. Signatures	FuncFormer
6s410	80.07	**92.89**
6s407	65.73	**70.37**
6s264	60.54	**69.59**
6s380	50.93	**93.02**
6s413	63.41	**79.56**
Average	64.14	**81.09**

Note: Entries are accuracy (%). Level-GNN uses a standard aggregation backbone; FuncFormer uses the proposed dual-path attention backbone.

B Additional controlled experiments

Table 12: Controlled input-feature comparison on the functional-reasoning benchmark. DeepGate4+PI-signature replaces DeepGate4's usual input feature with the PI-driven binary signature $\mathbf{h}_v^0$ while retaining the DeepGate4 encoder; it does not receive FuncFormer's level-wise spatial or latch-conditioned temporal traces.

Circuit	DeepGate4	DeepGate4+PI-signature	FuncFormer
D1	94.25	91.63	96.24
D2	93.11	92.00	95.16
D3	92.92	90.09	95.03
D4	92.91	91.19	95.08
D5	83.44	80.96	85.66
D6	92.46	91.08	94.76
D7	82.62	79.70	84.96
D8	92.38	90.57	94.79
D9	92.36	91.28	94.83
D10	92.21	89.42	94.74
D11	92.08	90.63	94.68
D12	91.83	89.79	94.61
D13	91.74	90.59	94.56
D14	86.50	83.53	89.28
D15	91.51	89.65	94.35
D16	91.47	89.12	94.37
D17	91.28	89.96	94.24
D18	87.56	84.70	90.58
D19	91.10	89.00	94.18
D20	91.04	88.63	94.19
Mean accuracy	**90.74**	**88.68**	**93.31**

Entries are accuracy (%). This control tests the PI-driven signature as an unstructured input feature in the DeepGate4 pipeline. It does not test DeepGate4 with FuncFormer's full spatial and temporal trajectories.

Table 13: Sensitivity to temporal unrolling length on the five HWMCC-13 circuits used in property scheduling.

Circuit	$N = 5$	$N = 10$	$N = 15$
6s410	81.58	92.89	94.74
6s407	62.04	70.37	74.07
6s264	61.34	69.59	71.13
6s380	84.96	93.02	95.58
6s413	68.18	79.56	81.82
Average accuracy (%)	**71.62**	**81.09**	**83.47**

Entries are accuracy (%). The five circuits were selected to cover multiple ground-truth solver-runtime classes with a comparatively balanced difficulty distribution. These measurements describe this selected set; they do not establish that increasing N improves performance on all sequential designs.

Table 14 reports the functional-reasoning accuracy for each of 20 circuits under all five randomized-simulation seeds, together with each circuit's mean and population variance. Accuracy values are percentages; the variance is in percentage-point squared. The seed-wise macro-average accuracies are 93.31%, 93.33%, 93.45%, 93.15%, and 93.20%, giving a mean of 93.29% and sample standard deviation of 0.12 percentage points. Only the randomized simulation seed changes; the data split, model initialization, minibatch order, optimizer, hyperparameters, and other training randomness are fixed. Thus, the experiment measures sensitivity to signature sampling rather than full training-seed variability.

Table 14: Per-circuit functional-reasoning accuracy over five simulation seeds.

Circuit	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Mean	Variance
D1	96.24	95.88	96.15	96.02	95.95	96.048	0.017176
D2	95.16	95.28	95.54	94.85	95.02	95.170	0.054800
D3	95.03	94.81	95.18	95.42	94.95	95.078	0.043576
D4	95.08	95.31	95.52	94.75	94.90	95.112	0.076536
D5	85.66	85.42	85.91	85.35	85.78	85.624	0.044824
D6	94.76	94.92	95.18	94.45	94.61	94.784	0.063544
D7	84.96	85.38	85.12	84.62	84.80	84.976	0.068384
D8	94.79	94.42	94.65	94.28	94.51	94.530	0.031400
D9	94.83	95.05	95.31	94.48	94.68	94.870	0.083160
D10	94.74	95.08	94.51	94.88	94.62	94.766	0.039824
D11	94.68	94.35	94.52	94.18	94.41	94.428	0.027976
D12	94.61	95.05	94.80	94.25	94.42	94.626	0.078824
D13	94.56	94.20	94.72	94.38	94.95	94.562	0.067936
D14	89.28	89.12	89.65	89.41	88.95	89.282	0.057656
D15	94.35	94.52	94.78	93.95	94.18	94.356	0.080504
D16	94.37	94.12	93.95	94.28	94.05	94.154	0.023224
D17	94.24	94.65	94.40	93.82	94.08	94.238	0.079136
D18	90.58	90.75	91.02	90.88	90.65	90.776	0.025064
D19	94.18	94.35	93.82	94.61	94.02	94.196	0.073544
D20	94.19	93.85	94.32	94.08	94.56	94.200	0.056200
Macro-average	93.31	93.33	93.45	93.15	93.20	93.29	–

C Visualization protocol for Figure 1

For Figure 1, we obtain one representation per toy multiplier variant and project them to two dimensions with PCA. For the DeepGate2 comparison, we use the node embeddings produced by the trained DeepGate2 model. For FuncFormer, the functional view is the PI-driven signature at the target node, while the structural view is the mean of its projected embeddings across the K propagation layers. The resulting points are projected by PCA; the plotted axes are PC1 and PC2 and have no independent semantic interpretation.

D Proof of Theorem 3.1

Proof. Let $\mathcal{X} = \{0,1\}^n$ be the input space of two node functions $f_u, f_v : \mathcal{X} \to \{0,1\}$. For an input pattern $x \in \mathcal{X}$, define

$$Y(x) = \mathbf{1}(f_u(x) \neq f_v(x)).$$

The functional disagreement probability under the uniform distribution is

$$p = \Pr_{x \sim \mathcal{U}(\mathcal{X})}[f_u(x) \neq f_v(x)] = \frac{1}{2^n} \sum_{x \in \mathcal{X}} Y(x).$$

FuncFormer samples d_{sim} input patterns $x_1, \ldots, x_{d_{sim}}$ uniformly and independently, i.e., with replacement. The sampled binary signatures satisfy $\mathbf{h}_u[k] = f_u(x_k)$ and $\mathbf{h}_v[k] = f_v(x_k)$. Since the signatures are binary,

$$\frac{1}{d_{sim}} \|\mathbf{h}_u - \mathbf{h}_v\|_2^2 = \frac{1}{d_{sim}} \sum_{k=1}^{d_{sim}} \mathbf{1}(\mathbf{h}_u[k] \neq \mathbf{h}_v[k]) = \frac{1}{d_{sim}} \sum_{k=1}^{d_{sim}} Y(x_k).$$

Each $Y(x_k)$ is a Bernoulli random variable with mean p. Therefore,

$$\mathbb{E}\left[\frac{1}{d_{sim}} \|\mathbf{h}_u - \mathbf{h}_v\|_2^2\right] = \frac{1}{d_{sim}} \sum_{k=1}^{d_{sim}} \mathbb{E}[Y(x_k)] = p. \tag{6}$$

This proves unbiasedness for the normalized disagreement probability. If one instead wants the unnormalized truth-table Hamming distance $H_{true} = 2^n p$, then the corresponding unbiased estimator is $2^n d_{sim}^{-1} \sum_k Y(x_k)$.

For variance, independence gives

$$\mathrm{Var}\left(\frac{1}{d_{sim}} \sum_{k=1}^{d_{sim}} Y(x_k)\right) = \frac{1}{d_{sim}^2} \sum_{k=1}^{d_{sim}} \mathrm{Var}(Y(x_k)) = \frac{p(1-p)}{d_{sim}} \leq \frac{1}{4 d_{sim}}. \tag{7}$$

Thus, the estimator variance decreases as $O(1/d_{sim})$. Since $Y(x_k) \in [0,1]$ are independent bounded variables, Hoeffding's inequality further gives

$$\Pr\left(\left|\frac{1}{d_{sim}}\|\mathbf{h}_u - \mathbf{h}_v\|_2^2 - p\right| \geq \epsilon\right) \leq 2\exp(-2d_{sim}\epsilon^2). \tag{8}$$

This is the concentration result used in the main text. □

Remark on non-uniform sequential states. The theorem estimates disagreement under the sampling distribution used to generate the signatures. For combinational circuits with uniform random PIs, this is the uniform truth-table disagreement probability. In sequential circuits, latch states induced by real traces may follow a non-uniform reachable-state distribution. Under such a distribution, the same estimator remains unbiased for the disagreement probability of the sampled distribution, but not necessarily for the uniform full-state truth table. In sequential circuits, this analysis describes disagreement under the sampled state distribution.

Scope of the theorem. The result is a property of the signature-generation process. If the sampled patterns are drawn uniformly, the normalized signature distance estimates uniform Boolean disagreement. If the sampler is replaced by a guided or reachable-state-aware distribution, the same argument estimates disagreement under that distribution, provided the samples are independent. It describes the information contained in the functional signatures before task-specific learning.

E Additional implementation details

Circuit and verification terminology. An And-Inverter Graph (AIG) is a technology-independent representation whose internal nodes are two-input AND gates; complemented edges encode inversion, and primary inputs and constants form the leaves. Other Boolean gates are lowered to this representation before simulation. A formal-verification property is a Boolean assertion over a transition system, optionally under an initial-state condition and environment constraints. Within the common solver budget, a property is counted as solved if the exact checker returns either a proof or a counterexample; timeout, memory exhaustion, or an unknown result is counted as unresolved. The learned score only determines scheduling order.

Architecture. Both the spatial and temporal encoders use four transformer layers, eight attention heads, and GELU activations in the feed-forward networks. The default simulation width is $d_{sim} = 256$, and all circuits use temporal unrolling length $N = 10$. Spatial traces are variable-length sequences, so mini-batch construction pads them to the batch maximum length and supplies attention masks to exclude padded entries. The spatial branch summary is obtained by masked mean readout over valid trace entries, and the temporal branch uses the same readout form over the N valid time steps. Spatial and temporal outputs are concatenated and passed to the MLP readout defined in Section 3.3.

Sequential preprocessing and baselines. FuncFormer represents sequential circuits through the latch-conditioned expansion described in Section 3.1. Existing baselines such as GCN, GAT, and DeepGate4 operate on the original graph with their native message-passing or attention mechanisms and do not receive FuncFormer's latch-conditioned temporal traces. All methods use the same labels, train/test splits, and downstream heads whenever possible, but the sequential comparison is therefore between complete representation pipelines, not a matched encoder-only ablation. A controlled DeepGate4+signature comparison on the combinational functional-reasoning task is reported in Appendix B; it does not remove this sequential-protocol difference.

Optimization. All models are trained with AdamW using learning rate 10^{-4}, weight decay 10^{-5}, and batch size 512. We use the same downstream heads as the corresponding baselines whenever possible, so that the comparison focuses on the representation module.

Compute resources. The implementation uses PyTorch Geometric 2.5.3. The standard experiments use an enterprise Linux server equipped with an Intel Xeon Gold 6246R CPU and a GeForce RTX 3090 Ti GPU. The additional OpenABC-D training and largest-circuit timing run used three GPUs and was reported as an RTX 3060 setup with 24 GB memory. Measurements from these settings are kept separate in the efficiency table.

Existing assets and licenses. We use public academic benchmarks and tools for evaluation, including OpenABC-D, ITC'99, IWLS'05, OpenCores-derived designs, HWMCC-13, ABC, and PyTorch Geometric, and cite their original sources in the main paper. PyTorch Geometric is released under the MIT license, and ABC is distributed under a BSD-style license with bundled components under their own licenses.

Loss functions. For QoR prediction, we minimize mean squared error, $\mathcal{L}_{QoR} = |\mathcal{D}|^{-1} \sum_{i \in \mathcal{D}} (\hat{y}_i - y_i)^2$. For functional reasoning, we use a contrastive objective following DeepGate2: equivalent node pairs are pulled together by maximizing cosine similarity, while non-equivalent pairs are separated by a margin. For verification-hardness prediction, labels are generated from exact-solver runtimes measured in seconds: we discretize runtimes by $\lfloor \log_{10}(\text{time (s)}) \rfloor$ and optimize cross-entropy over the resulting difficulty classes. The predicted classes are then used only to rank properties for scheduling; final verification outcomes remain produced by the exact solver.

Metric aggregation and scheduling. For QoR prediction, for each held-out design i we compute $r_i = \mathrm{MSE}_{\mathrm{FuncFormer},i} / \mathrm{MSE}_{\mathrm{DeepGate4},i}$ and report the unweighted mean of r_i. The main-table value 0.74 therefore corresponds to a 26.29% mean per-design reduction; this differs from a ratio of pooled MSEs. In Table 2, each "Impr." column instead reports reduction relative to GCN, as stated in that table's note. For functional reasoning and hardness prediction, we report accuracy and use absolute percentage-point differences when comparing to DeepGate4. For property scheduling, each property is assigned a predicted hardness score and properties are ordered from easier to harder under the learned ranking. The solver is then run under the same global budget for all methods. We report aggregate solved counts as the primary scheduling metric and state ratios against their named reference methods.

F Overall algorithm flow and time complexity

Overall flow. FuncFormer proceeds in six stages. First, the input netlist is represented as an AIG; for sequential circuits, latch-conditioned expansion splits each latch into current-state pseudo-input and next-state pseudo-output nodes and forms N time frames. Second, functional signatures of width d_{sim} are assigned to PIs, and for spatial level-wise propagation the nodes at each starting level j are re-initialized as pseudo-PIs with fresh signatures. Third, signatures are propagated in topological order over the actual fanin connections using AIG operations: AND is element-wise multiplication and NOT is element-wise complement. Fourth, for each node v^i, spatial traces are organized as $\mathbf{H}_{spa}^{v^i} = \{\mathbf{h}_{v^i}^0, \ldots, \mathbf{h}_{v^i}^i\}$, while temporal traces are organized as $\mathbf{H}_{tem}^{v^i} = \{\mathbf{h}_{v^i}^0(\mathcal{L}_0), \ldots, \mathbf{h}_{v^i}^0(\mathcal{L}_{N-1})\}$. Fifth, the binary signatures are projected into continuous embeddings and encoded by the spatial and temporal Transformer branches. Finally, the two branch outputs are concatenated and passed to an MLP before the downstream task heads.

Time complexity. Let $|V|$ be the number of gates, N the temporal unrolling steps, d_{sim} the simulation width, D the maximum spatial logic depth, and d_{model} the latent dimension. With cached level-wise propagation, FuncFormer visits each node once in topological order and stores the traces associated with different starting levels. If L_v denotes the number of spatial traces retained for node v, the spatial Boolean-propagation cost is

$$O\left(d_{sim} \sum_{v \in V} L_v\right),$$

which is upper-bounded by $O(|V| D d_{sim})$ when $L_v \leq D$. Temporal Boolean propagation over N latch-conditioned frames costs $O(N d_{sim} |V|)$. The spatial encoder attends over the local logic-depth trajectory with $O(D^2 d_{model})$ cost per node, while the temporal encoder attends over the N-step trajectory with $O(N^2 d_{model})$ cost per node. Thus, a conservative upper bound is

$$\mathcal{T}_{total} = O\big(|V|\big((D+N) d_{sim} + D^2 d_{model} + N^2 d_{model}\big)\big).$$

Caching reduces redundant graph traversal, but the work and memory still scale with the number of traces retained per node. The relevant design question is the trade-off between functional fidelity and memory, controlled by d_{sim}, D, N, and mini-batch size.

Empirical efficiency and scale. On ITC'99, Table 15 shows that FuncFormer uses 16.72 GB of training memory and 1.72 s inference time, compared with 7.53 GB and 1.46 s for DeepGate4. The additional memory reflects storage of simulation trajectories; the hardware setup is given in Appendix E. Table 16 reports the largest measured OpenABC-D test circuit, `jpeg` (235,573 nodes): FuncFormer takes 5.44 s for preprocessing and 10.6 s for inference, with 8.93 GB peak memory, while DeepGate4 takes 12.9 s for inference and uses 5.37 GB; its preprocessing time was not reported. FuncFormer's inference time excludes K-hop preprocessing. The separate OpenABC-D training run used three GPUs for 1.97 h total (5.91 GPU-hours) in the reported RTX 3060 setup with 24 GB memory. For scale context, HWMCC-13 circuit `6s410` has 270,052 nodes; its measurements use a different benchmark and run configuration from the ITC'99 comparison and should not be compared directly.

Table 15: Training memory and inference time on ITC'99.

Method	**Training Memory**	**Inference Time**
DeepGate4	7.53 GB	1.46 s
FuncFormer	16.72 GB	1.72 s

Table 16: Runtime and memory on the largest reported OpenABC-D test circuit.

Circuit (nodes)	**Method**	**Preprocessing**	**Inference**	**Peak memory**
`jpeg` (235,573)	FuncFormer	5.44 s	10.6 s	8.93 GB
	DeepGate4	-	12.9 s	5.37 GB

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 abstract and introduction state the functional-signature representation, dual-path transformer, estimator analysis, and three evaluated EDA tasks. The claims match the theoretical result, experiments, and stated scope.

   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: The paper includes a dedicated Conclusion and Limitations section discussing the use of FuncFormer as a learned ranking signal rather than a correctness certificate, as well as tunable resource costs controlled by simulation width, spatial logic depth, and temporal unrolling length. The appendix further discusses the sampling distribution assumed by the estimator.

   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: [Yes]

   Justification: The theorem states the sampling assumptions, provides a proof sketch in the main text, and gives the full derivation with discussion of non-uniform reachable-state distributions in Appendix D.

   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: The paper reports datasets, train/test splits, architecture, optimization settings, loss functions, metric aggregation, scheduling protocol, and full per-circuit results in the main text and appendices.

   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: [Yes]

   Justification: We provide our main codes at `https://github.com/MIRALab-USTC/FuncFormer.git`.

   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: The paper specifies datasets, splits, baselines, architecture, optimizer, learning rate, batch size, losses, metric aggregation, and scheduling details in Section 4 and Appendix E.

   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: [No]

   Justification: The appendix reports five randomized-simulation seeds for functional reasoning (mean accuracy 93.29%, sample SD 0.12 percentage points), varying only the signature-sampling seed. The QoR, verification-hardness, and scheduling results do not report run-to-run uncertainty or significance tests, and the reported standard deviation does not capture training-seed variability.

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: [Yes]

   Justification: Appendix E reports the implementation framework, CPU/GPU hardware, optimizer configuration, and batch size. Appendix F further reports complexity analysis together with peak training memory and inference time measurements.

   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 public circuit benchmarks and does not involve human subjects, personal data, or privacy-sensitive data. The intended use is EDA representation learning and verification support.

   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 work targets circuit representation learning and verification scheduling. It does not involve human subjects, personal data, surveillance, or generative misuse pathways. In safety-critical verification workflows, the learned model is used only for ranking, while exact solvers retain correctness certification.

    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, scraped datasets, generative models, or assets with substantial misuse risk requiring safeguards.

    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: The paper cites the original sources for the benchmark datasets, baseline methods, and verification tools used in the evaluation. Appendix E describes the licensing status of the main software assets.

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: [N/A]

Justification: The paper does not release new assets.

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, participant instructions, compensation, or human-subject data.

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 human subjects, so IRB or equivalent approval 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: LLMs are not used as an important, original, or non-standard component of the core method, experiments, data processing, or scientific analysis.

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.