NetTAG: A Multimodal RTL-and-Layout-Aligned Netlist Foundation Model via Text-Attributed Graph

Wenji Fang[1], Wenkai Li[1], Shang Liu[1], Yao Lu[1], Hongce Zhang[2], Zhiyao Xie[1*]
[1]Hong Kong University of Science and Technology
[2]Hong Kong University of Science and Technology (Guangzhou)
[*]Corresponding Author

***Abstract*—Circuit representation learning has shown promise in advancing Electronic Design Automation (EDA) by capturing structural and functional circuit properties for various tasks. Existing pre-trained solutions rely on graph learning with complex functional supervision, such as truth table simulation. However, they only handle simple and-inverter graphs (AIGs), struggling to fully encode other complex gate functionalities. While large language models (LLMs) excel at functional understanding, they lack the structural awareness for flattened netlists. To advance netlist representation learning, we present NetTAG, a netlist foundation model that fuses gate semantics with graph structure, handling diverse gate types and supporting a variety of functional and physical tasks. Moving beyond existing graph-only methods, NetTAG formulates netlists as text-attributed graphs, with gates annotated by symbolic logic expressions and physical characteristics as text attributes. Its multimodal architecture combines an LLM-based text encoder for gate semantics and a graph transformer for global structure. Pre-trained with gate and graph self-supervised objectives and aligned with RTL and layout stages, NetTAG captures comprehensive circuit intrinsics. Experimental results show that NetTAG consistently outperforms each task-specific method on four largely different functional and physical tasks and surpasses state-of-the-art AIG encoders, demonstrating its versatility.**

I. Introduction

Machine learning (ML) techniques have demonstrated remarkable achievements in electronic design automation (EDA). Existing ML methods are mostly tailored to specific tasks such as predicting physical metrics (e.g, timing [1], [2], [3], area [4], [5], [6], power [7], [8], [9], and routability [10], [11], [12]) or the reasoning of circuit functionalities [13], [14], [15], [16]. These methods are typically developed by supervised training, requiring extensive label collection and model customization for each task. Despite obvious effectiveness, they are time-consuming to develop and lack generalizability, often capturing only task-specific patterns rather than a generalizable understanding of circuits.

Recent circuit representation learning methods (i.e., encoders) generate informative embeddings for circuits and are able to support a range of downstream tasks [17], [18]. Existing methods mainly focus on VLSI circuit netlists, and some works also explore RTL circuits [19], analog circuits [20], and FPGAs [21]. We provide a detailed summary of the netlist encoders in TABLE I. All these netlist encoders capture circuit functionality through graph structure leveraging graph learning models, such as Graph Neural Networks (GNNs) or Graph Transformers (GTs). They either directly infer functionality with graph topology [22], [23], [24], or employ functional information to pre-train the models [25], [26], [27], [28], [29], [30], [31].

However, we argue that existing graph-only netlist encoders lack sufficient expressiveness to capture complex circuits. **These methods rely on graph models that inherently prioritize *structural* information over *semantics* (i.e., functionality).** Their limitations include: *(1) Limited to and-inverter graph (AIG).* Most methods target only simple AIG-based netlists, representing a narrow subset of standard cell libraries, and cannot handle post-mapping netlists with diverse gate types. *(2) Dependence on complex functional supervision.* Representative methods like the DeepGate family [25], [26], [27], [30] and FGNN [28], [29] incorporate truth table simulation to capture AIG functionality. However, truth tables suffer from exponential state expansion when applied to post-synthesis netlists with multi-input complex gates (e.g., full adders, multiplexers), limiting their applicability. *(3) Lack of physical information.* None of these methods consider physical characteristics, focusing solely on logic functionality. Consequently, these encoders are limited to AIG-based tasks (e.g., equivalence checking, SAT solving) and struggle with tasks for post-mapping netlists [2], [3], [7], [10], [11], [12], [14], [15], especially those evaluating physical design quality.

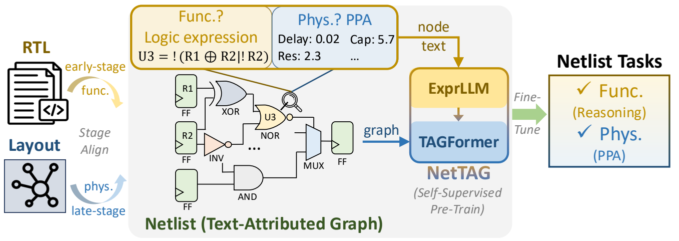


Fig. 1: Overview of NetTAG. Netlists are formulated as text-attributed graphs, with functional and physical text attributes extracted for each gate. Within NetTAG, gate attributes are initially encoded by ExprLLM, then refined with global netlist graph structures using TAGFormer. NetTAG is pre-trained with self-supervised objectives and aligned with RTL and layout embeddings, enabling versatile support for both functional and physical tasks after fine-tuning.

Recently, large language models (LLMs) have demonstrated remarkable expressiveness in circuit-related generative tasks [32], [33], [34]. **However, LLMs inherently capture circuit textual *semantics* rather than *structure*.** Limited work has explored LLMs for general-purpose circuit representation learning, with key limitations summarized as follows: *(1) Struggle with netlists.* Although LLMs can interpret circuit functionality from RTL code, the gate-level netlists are more flattened and lack informative context, making functional understanding more challenging. *(2) Lack of structural encoding.* LLMs struggle to capture the circuit structures, limiting their utility for netlist representation learning.

In this work, we present **`NetTAG`**, a foundation model[1] for netlists that captures functional and physical properties across diverse gate types. Unlike previous circuit encoders and LLM solutions focusing on single circuit modality, NetTAG fuses gate text *semantics* with global graph *structures* to achieve *functional* and *physical* understanding. Serving as a foundation model, the pre-trained NetTAG generates versatile embeddings for logic gates, register cones, and full circuits. The embeddings can be easily fine-tuned for various downstream tasks, supporting largely different netlist-stage functional and physical tasks across these circuit granularities. As shown in Fig. 1, we propose the following innovative strategies in NetTAG:

- **Preprocess: formulating netlists as text-attributed graphs.** This paper is the first to represent netlists in text-attributed graphs (TAGs) format. In the netlist graph, we annotate each gate with functional symbolic logic expressions and physical characteristics as node-level text attributes. Our TAG format combines gate textual attributes with graph topology, moving beyond existing graph-only circuit representation learning.

[1]The code and pre-trained NetTAG model are available at https://github.com/hkust-zhiyao/NetTAG. The pre-trained model enables users to easily generate and fine-tune embeddings for their own netlist tasks.

TABLE I: Comparision of state-of-the-art netlist representation learning methods.

Method	Target Circuit		Encoding Methodology				Downstream Tasks	
	Cell Type	Circuit Type	Modality	ML Model	Pre-Train Target	Cross-Stage Align	Target	Type
DeepGate1/2 [25], [26]	AIG	Comb.	Graph	GNN	Gate	N/A	Gate	Func.
DeepGate3 [27]	AIG	Comb.	Graph	GT	Gate&Circuit	N/A	Gate	Func.
FGNN [28], [29]	AIG	Comb.	Graph	GNN	Circuit	N/A	Gate&Circuit	Func.
HOGA [23]	AIG	Comb.&Seq.	Graph	GNN	N/A	N/A	Gate&Circuit	Func.&Phys.
NetTAG (Ours)	**Any Gate**	**Comb.& Seq.**	**Text& Graph**	**LLM& GT**	**Gate& Reg. Cone**	**RTL& Layout**	**Gate& Reg. Cone& Circuit**	**Func.& Phys.**

- **Multimodal architecture: fusing semantic and structure.** Leveraging TAGs, NetTAG introduces an innovative multimodal architecture that combines an LLM-based text encoder (i.e., ExprLLM) with a graph transformer (i.e., TAGFormer). ExprLLM first encodes *fine-grained* gate text attributes into semantic-rich node embeddings, then TAGFormer refines them with *overall* graph structure, generating final individual gate and overall graph embeddings.
- **Aligned pre-training: self-supervised and cross-stage-aware.** We introduce four circuit-specific self-supervised objectives to pre-train NetTAG across circuit granularities, enhancing Boolean logic understanding and fusing gate semantics with the global graph structure. Moreover, we align netlist embeddings with those from RTL and layout stages to strengthen cross-stage functional and physical awareness.

We evaluate NetTAG on four largely different EDA tasks, covering both functional reasoning and physical design quality predictions at different circuit granularities. NetTAG consistently outperforms all state-of-the-art (SOTA) task-specific models, achieving 14% and 13% higher accuracy in logic gate and register function identification, respectively, and reducing errors by 2% for register slack, and 7% for both circuit area and power predictions. NetTAG demonstrates strong generalization across these representative tasks. Finally, we evaluate the scalability of NetTAG by scaling up model size and dataset volume. The results indicate a huge potential for improvement by scaling up either model or circuit datasets in future work.

II. Methodology

A. Overview

Fig. 2 outlines the NetTAG workflow. First, netlist data is converted into TAG format, detailed in Section II-B. NetTAG's multimodal architecture, which combines fine-grained gate text and overall circuit graph refinement, is illustrated in Section II-C. Step 1 enhances Boolean logic comprehension for ExprLLM (Section II-D), and Step 2 fuses gate semantics with circuit structure in TAGFormer, incorporating cross-stage alignment to strengthen both functional and physical understanding (Section II-E). Finally, Section II-F demonstrates the fine-tuning strategy that leverages the pre-trained NetTAG for various downstream tasks.

B. Circuit Data Preprocessing

Conventional graph methods like GNNs and GTs mainly capture structural information but fall short when the semantic context is also needed. TAG representation learning bridges this gap by encoding nodes as text and leveraging graph connections [35], [36], effectively improving node classification performance. *Intuitively, netlists, with their graph structure and detailed gate logic functions, are ideally suited for formulation as TAGs.*

Formulating netlist as TAG. Given a gate-level netlist $\mathcal{N}$, we represent it as a text-attributed graph $\mathcal{G}_{\mathcal{N}} = \{\mathcal{T}, \mathcal{E}\}$, where each gate in the netlist is a node in the graph. Here, $\mathcal{T} = \{t_1, \ldots, t_m\}$ is a set of gate text attributes, where m representing the total gate count. Each attribute t_i contains functional and physical properties of the i^{th} gate, as shown in Fig. 3(b), and consists of a sequence of tokens t_i. $\mathcal{E}$ is the set of edges connecting the gates.

Specifically, for functionality, we extract the symbolic logic expression for each gate based on its k-hop fan-in cone. These expressions are constructed using a formal verification toolkit [37], providing formal definitions for gate functionality. An example is demonstrated in Fig. 3(b), the 2-hop expression for this NOR gate with the symbolic name "U3" is derived from the Boolean functions of nodes within its 2-hop fan-in cone. The expression for "U3" is represented as: "U3 = !((R1 ⊕ R2)|!R2)". Each expression includes symbolic names for input gates and the Boolean operations. We summarize the key advantages of exploiting the symbolic expression for functional encoding:

1) **Versatile gate support:** Our symbolic expressions use Boolean formulas to represent any gate type, including complex gates like AOI and full adders. This enables NetTAG to support diverse gate types in netlists, beyond the limitations of AIG formats.
2) **Direct functional encoding:** In contrast to pre-trained circuit encoders that rely on graph-based learning with truth table supervision [29], [27], symbolic expressions enable straightforward static analysis, covering all input conditions without exponential growth problems by exhaustive truth table simulation.
3) **Structure independence:** Unlike structure-based methods that infer functionality with graph topology [23], [24], our approach derives gate expressions from Boolean formulas. It easily recognizes the same functionality across varying structures and distinguishes different functions in similar topologies.
4) **Compatible with language models:** Expressions can be seamlessly processed by language models, unleashing LLMs' reasoning capabilities to enhance circuit functional understanding. This functional expression also enables self-supervised pre-training for LLMs, such as expression contrastive learning, to improve LLMs' logic understanding from expression data itself.

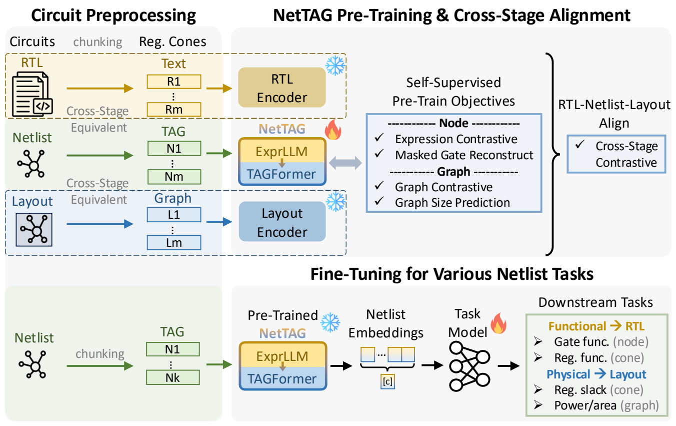


Fig. 2: NetTAG Workflow. Sequential netlists are chunked into combinational register cones and converted into TAGs. During pre-training, NetTAG is trained with node-level and graph-level self-supervised objectives, and it is aligned with RTL and layout embeddings. The pre-trained NetTAG then generates netlist embeddings, which are fine-tuned with netlist-stage task labels.

For the physical characteristics, we annotate each gate with information extracted from the standard cell library, including characteristics such as power, area, delay, toggle rate, probability, load, capacitance, and resistance. Both functional and physical attributes are combined to form the text attribute for each gate.

Processing RTL and layout data. In addition to the target netlists, we utilize corresponding RTL and layout data to enhance NetTAG's cross-stage awareness. As shown in Fig. 3(a) and (c), RTL code is processed directly as text containing functional semantics, while layout data is represented as connectivity graphs annotated with physical characteristics. Specifically, nodes in the layout graphs are

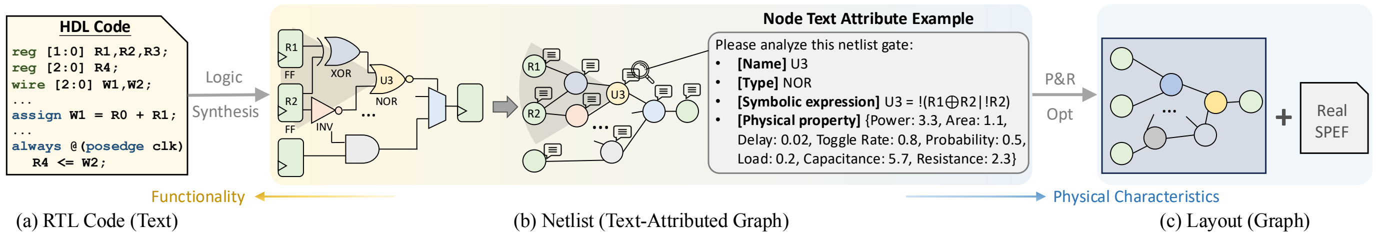

HDL Code
reg [1:0] R1,R2,R3;
reg [2:0] R4;
wire [2:0] W1,W2;
...
assign W1 = R0 + R1;
...
always @(posedge clk)
R4 <= W2;
Logic Synthesis
Node Text Attribute Example
Please analyze this netlist gate:
[Name] U3
[Type] NOR
[Symbolic expression] U3 = !(R1⊕R2|!R2)
[Physical property] {Power: 3.3, Area: 1.1, Delay: 0.02, Toggle Rate: 0.8, Probability: 0.5, Load: 0.2, Capacitance: 5.7, Resistance: 2.3}
P&R Opt
Real SPEF
Functionality
Physical Characteristics
(a) RTL Code (Text)
(b) Netlist (Text-Attributed Graph)
(c) Layout (Graph)

Fig. 3: Circuit data design stages and modalities. RTL code is processed directly as text. Netlists are represented as TAGs, with node attributes including gate name, type, symbolic expression, and physical property. Symbolic expressions are derived from each gate's k-hop input cone. Layout data is converted into graphs and annotated with physical information extracted from the SPEF file.

annotated with capacitance, resistance, and delay values extracted from the Standard Parasitic Extraction Format (SPEF) file.

Chunking sequential circuit into register cones. To handle large-scale sequential circuits, we propose chunking the original circuit into register cones. For each register, we backtrace through all driving logic up to other registers, creating a subcircuit cone that captures the complete state transition of the register, including update relationships and timing paths. This chunking process significantly reduces circuit size for our NetTAG, enhancing scalability. We apply this method consistently across netlist, RTL, and layout data, ensuring that cross-stage cones remain functionally equivalent[2] for subsequent alignment.

C. NetTAG Model Architecture

Fig. 4 illustrates the architecture of NetTAG, which comprises two main components: ExprLLM for gate-level text encoding and TAGFormer for further circuit-level graph encoding. Additionally, two auxiliary pre-trained encoders are employed to generate embeddings for the RTL code and layout graph, facilitating cross-stage alignment. We detail our NetTAG architecture below.

ExprLLM of NetTAG is initialized using an LLM-based text embedding model to encode gate text attributes as initial node embeddings. Following [38], we adapt a decoder-only LLM into a general-purpose text encoder by converting causal attention to bidirectional attention, which has shown superior encoding performance over encoder-only models (e.g., BERT) [39]. ExprLLM takes the gate text attributes t_i as input, converting them into the text embeddings T_i:

$$T_i = \text{ExprLLM}(t_i). \tag{1}$$

TAGFormer of NetTAG refines node embeddings from ExprLLM using a graph transformer model, which captures netlist graph structure via global attention. We include a `[CLS]` node connected to all other nodes in the graph, serving as the graph-level embedding. The initial embedding for each gate n_i is created by concatenating its text embedding T_i from ExprLLM with its physical characteristics vector $x_{\text{phys}i}$. TAGFormer then processes node embeddings $\{n_1, \dots, n_m\}$ with their connectivity $\mathcal{E}$ into a sequence of netlist gate embeddings $\{N_1, \dots, N_m, N_{\text{cls}}\}$, where N_{cls} represents the entire graph embedding:

$$\begin{aligned} n_i &= (T_i, x_{\text{phys}_i}), \\ \{N_1, \dots, N_m, N_{\text{cls}}\} &= \text{TAGFormer}(\{n_1, \dots, n_m\}, \mathcal{E}). \end{aligned} \tag{2}$$

Auxiliary RTL and Layout Encoders support cross-stage alignment for NetTAG by providing functional and physical insights from RTL and layout stages. The pre-trained RTL encoder, an LLM-based text encoder, generates RTL embeddings R_{cls}. Similarly, the layout encoder, a pre-trained graph transformer, produces layout embeddings L_{cls}. Please note that these auxiliary cross-stage encoders are used only during pre-training.

D. Pre-Training Step 1: Enhacing logic understanding in ExprLLM

We design pre-training objectives tailored for circuit netlists to capture both functional and structural information using our multimodal NetTAG. As shown in Fig. 4, the pre-training process is divided into two main steps: (1) Pre-training ExprLLM to strengthen its understanding of Boolean logic, and (2) Pre-training TAGFormer to fuse gate text semantics from frozen ExprLLM with circuit graph structure. Additionally, two pre-trained RTL and layout encoders are used for cross-stage alignment. We detail our pre-training objectives below.

Objective #1 Symbolic expression contrastive learning. In step1, we apply logic expression contrastive learning to enhance ExprLLM's understanding of Boolean expressions. As shown in Fig. 3, we begin by building a gate expression dataset from 2-hop symbolic expressions[3]. Then each expression is transformed using randomly applied Boolean equivalence rules[4] to generate a new positive sample. The InfoNCE loss [40] is then applied to differentiate each positive expression pair from other negatives, formulated as:

$$\mathcal{L}_{\text{expr}}^{\#1} = \mathcal{F}_{\text{CL}}(\boldsymbol{T}, T^+) = -\log \frac{\exp(T_{\text{ori}} \cdot T^+/\tau)}{\sum_{i=0}^{k} \exp(T_{\text{ori}} \cdot T_i/\tau)}, \tag{3}$$

where τ is the temperature scaling factor, T^+ is the positive embeddings, and $\boldsymbol{T}$ represents all k samples in the batch, including the original sample embeddings T_{ori} along with all other $(k-1)$ negative sample embeddings $T_i (i \neq \text{ori})$.

E. Pre-Training Step 2: Fusion in TAGFormer & Cross-Stage Align

1) Pre-training within TAGFormer for semantic and structure fusion: We propose gate-level and graph-level self-supervised objectives to jointly pre-train TAGFormer, fusing semantic-rich gate embeddings from pre-trained and frozen ExprLLM with the global circuit structure. The pre-training objectives are illustrated below:

Objective #2.1 Masked gate reconstruction. This objective aims to capture the structural roles of various gates by leveraging their connectivity within the netlist graph. Specifically, we propose to randomly mask a subset of gates in the netlist using a special `[MASK]` node. The model is then tasked with predicting the gate type (e.g., NOR, MUX, AND, etc.) of the masked gates based on the remaining unmasked nodes.

This objective is formulated as a multi-class classification problem, where the predicted gate types are treated as discrete labels. NetTAG first processes the masked graph to generate inputs and generates masked node embeddings N_i^{mask} for each masked gate in $\mathcal{T}^{\text{mask}}$. Then a classification Multi-layer Perceptron, denoted as $\text{MLP}_{\text{class}}$, takes the masked node embeddings N_i^{mask} as input and outputs the predicted probability distribution over gate types. The ground-truth gate type for each masked node i is denoted by $\boldsymbol{y}_i^{\text{mask}}$. The classification loss is defined using the cross-entropy loss as follows:

$$\mathcal{L}_{\text{gate}}^{\#2} = -\sum_{i \in \mathcal{T}^{\text{mask}}} \boldsymbol{y}_i^{\text{mask}} \log \left(\text{MLP}_{\text{class}}(N_i^{\text{mask}}) \right). \tag{4}$$

Objective #2.2 Netlist graph contrastive learning. This objective captures global structural and functional information by clustering similar netlists and separating dissimilar ones. Positive samples are generated via functionally equivalent transformations of each netlist graph [29], while negative samples are distinct graphs within the same batch. The InfoNCE loss is also applied to minimize the

[2]Most registers maintain unchanged throughout design stages.

[3]We choose 2-hop to balance the expression expansion and runtime.

[4]We implemented Boolean transformation rules such as De-Morgan's law, distributive law, commutative law, associative law, etc.

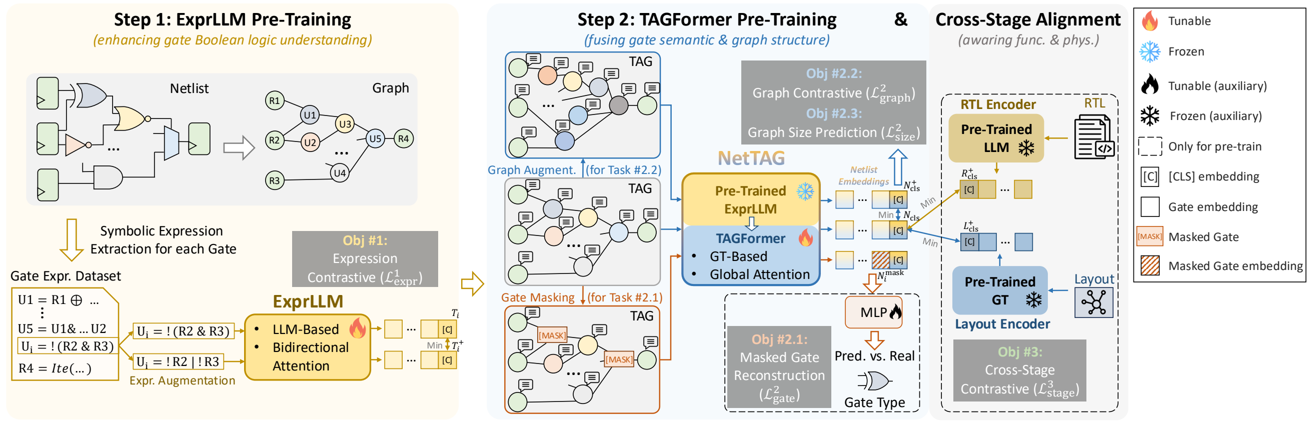


Fig. 4: NetTAG architecture and pre-training workflow. In step 1, we collect a dataset of gate symbolic expressions and pre-train ExprLLM to enhance its understanding of Boolean formulas. In step 2, ExprLLM is frozen, and we pre-train TAGFormer to fuse the gate semantics with the global graph structure. NetTAG embeddings are cross-stage aligned with those from pre-trained auxiliary RTL and layout encoders.

distance between positive pairs while maximizing separation from negative samples:

$$\mathcal{L}^{\#2}_{\text{graph}} = \mathcal{F}_{\text{CL}}(\boldsymbol{N}_{\text{cls}}, N^+_{\text{cls}}), \tag{5}$$

where N^+_{cls} denotes the positive sample embeddings, and $\boldsymbol{N}_{\text{cls}}$ includes the original and all other negative samples in the batch.

Objective #2.3 Netlist graph size prediction. This objective aims to predict the number of each type of gate in a given netlist graph based on the graph-level embedding, which is formulated as a regression problem. Denote $\boldsymbol{y}^{size}$ as the ground-truth gate counts for a netlist graph $\mathcal{G}_{\mathcal{N}}$. We employ an auxiliary model MLP_{regr}, which takes the graph-level embedding N_{cls} as input and outputs the predicted gate counts. The objective is to minimize the Mean Squared Error (MSE) between the predicted and actual gate counts, as formulated below:

$$\mathcal{L}^{\#2}_{\text{size}} = \frac{1}{n}\sum_{i=1}^{n}\left(\boldsymbol{y}_i^{size} - \text{MLP}_{\text{regr}}(N_{\text{cls}})\right)^2, \tag{6}$$

where n is the number of gate types, and each element in $\boldsymbol{y}^{size}$ corresponds to the predicted count for a specific gate type.

2) Pre-training beyond TAGFormer to enhance functional and physical awareness: **Objective #3 Cross-stage contrastive alignment.** Beyond the pre-training objectives within NetTAG's components, we introduce cross-stage alignment to integrate information from RTL-stage functionality and layout-stage physical implementation into NetTAG. This alignment enhances NetTAG's cross-stage awareness, benefiting both functional and physical downstream tasks, as further demonstrated in Section III-D. Using contrastive objectives, we align the circuit embeddings from RTL (i.e., R^+_{cls}), netlist (i.e., $\boldsymbol{N}_{\text{cls}}$), and layout (i.e., L^+_{cls}) within a shared latent space, as formulated below:

$$\mathcal{L}^{\#3}_{\text{align}} = \mathcal{F}_{\text{CL}}(\boldsymbol{N}_{\text{cls}}, R^+_{\text{cls}}) + \mathcal{F}_{\text{CL}}(\boldsymbol{N}_{\text{cls}}, L^+_{\text{cls}}). \tag{7}$$

To this end, we formulate the overall self-supervised pre-training objective of NetTAG as the following two-step process:

$$\mathcal{L}_{\text{NetTAG}} = \begin{cases} \mathcal{L}^1_{\text{expr}} & \text{(Step 1)} \\ \mathcal{L}^2_{\text{gate}} + \mathcal{L}^2_{\text{graph}} + \mathcal{L}^2_{\text{size}} + \mathcal{L}^3_{\text{align}} & \text{(Step 2)} \end{cases} \tag{8}$$

F. Fine-tuning for downstream tasks

NetTAG generates multi-grained embeddings for netlist elements, including combinational gates, register cones, and entire circuits. For sequential circuits, circuit-level embeddings are computed by summing the embeddings of all register cones, while for combinational circuits, their overall embeddings are directly obtained on the `[CLS]` node without the need for chunking into cones.

For downstream tasks, we fine-tune these embeddings with lightweight task models like MLPs or tree-based models (e.g., XGBoost). NetTAG's versatile embeddings support both regression and classification across functional and physical tasks for netlists. For functional tasks, the fine-tuning model predicts early RTL-stage information on netlists, such as gate functions and register types. As for physical tasks, it predicts late layout-stage metrics like register endpoint slack, power, and area. We will provide the detailed evaluation results in Section III-B.

III. Experimental Results

A. Experimental Setup

Data preparation. We collect circuits for pre-training from various sources, including benchmark RTL code from ITC99 [41], OpenCores [42], Chipyard [43], and VexRiscv [44]. All RTL designs are synthesized into netlists using Synopsys Design Compiler using NanGate 45nm technology library and then undergo physical design with Cadence Innovus. Gate and design quality metrics and statistics are obtained using Synopsys PrimeTime. For symbolic logic expressions, we use PySMT [37], a symbolic reasoning toolkit for formal verification, to construct and manipulate the Boolean expressions.

The statistical details of our dataset are shown in TABLE II. For the expression dataset used by ExprLLM, we collected 313k original expressions, each augmented with functionally equivalent transformations, resulting in a total of 626k expressions. As for the circuit netlist data, we collected 100k subcircuit cones after chunking, which were also functionally augmented to reach 200k samples in total. Additionally, we included 10k aligned RTL and layout cones for cross-stage alignment. When applying NetTAG to downstream tasks, we directly use the original task datasets if available. Otherwise, we utilize designs from the aforementioned open-source benchmarks.

TABLE II: Statistics of circuit expression and netlist dataset.

Source	Gate Expression		Circuit Netlist	
	# Data	# Tokens (Avg.)	# Data	# Nodes (Avg.)
ITC99	47k	6,960	4k	1,025
OpenCores	76k	212	55k	173
Chipyard	109k	9,849	20k	2,813
VexRiscv	81k	5,289	21k	901
Total	**313k**	**5,810**	**100k**	**855**

Implementation details. For the implementation of NetTAG, we initialize ExprLLM with LLM2Vec [38], an open-source LLM-based encoder based on Meta Llama-3.1-8B [45], with 8k maximum input tokens. For the graph transformer backbone of TAGFormer, we adopt SGFormer [46]. Additionally, we use NV-Embed [47] (maximum 32k input tokens) as the pre-trained RTL text encoder, and the layout encoder is pre-trained using a graph contrastive objective on SGFormer. Each MLP contains three layers with a hidden dimension of 256, and the output dimension of NetTAG is set to 768. We pre-train ExprLLM using LoRA on the full dataset for one epoch, followed by 50 epochs of pre-training for TAGFormer. Afterward, fine-tuning is performed with lightweight models and task-specific labels. All experiments are conducted on 8 Nvidia 4090 GPUs and 4 Nvidia A6000 GPUs.

B. Performance on Various Downstream Tasks

We evaluate the pre-trained NetTAG on four representative netlist-stage tasks, including functional and physical tasks across combinational and sequential circuits. These tasks span multiple netlist granularities: combinational gates, register cones, and entire circuits, offering a thorough assessment of NetTAG's capabilities.

NetTAG consistently outperforms task-specific baselines in each task, demonstrating its ability to capture functional and physical circuit information. This highlights NetTAG's role as a foundation model that is capable of generating informative embeddings for netlists. Detailed evaluations for each task are as follows:

Task1: Combinational gate function identification. This task identifies the functional type of each netlist combinational gate (e.g., adder, multiplier, comparator) as described in the original RTL code. Accurate gate function identification is critical for applications such as reverse engineering, hardware security, and functional verification. We evaluate NetTAG on an open-source dataset from GNN-RE [14], ensuring no label-related text is included in the gate text attributes and using the same metrics for a fair comparison.

As shown in TABLE III, NetTAG significantly outperforms GNN-RE across all metrics, with an average improvement of 14% in accuracy (97% vs. 83%), 11% in precision, 14% in recall and F1-score. These improvements demonstrate NetTAG's robust ability to capture functional information on each gate, enabling superior generalization compared to the task-specific GNN-based method.

TABLE III: Performance comparison on Task1: combinational gate function identification.

Design	GNN-RE [14]				NetTAG			
	Acc. (%)	Prec. (%)	Recall (%)	F1 (%)	Acc. (%)	Prec. (%)	Recall (%)	F1 (%)
1	79	82	79	74	97	97	97	97
2	96	96	96	96	100	100	100	100
3	94	94	94	94	100	100	100	100
4	78	83	78	78	100	100	100	100
5	91	92	91	90	99	99	99	99
6	74	78	74	68	94	94	94	93
7	80	80	80	80	84	87	84	81
8	89	90	89	87	95	96	95	96
9	65	77	65	67	100	100	100	100
Avg.	83	86	83	82	**97**	**97**	**97**	**96**

Task2: Sequential state/data register identification. This task distinguishes state registers from data path registers, which is essential for understanding high-level control and data flow within low-level netlists, supporting security analysis and verification. We evaluate NetTAG against ReIGNN [15], a supervised method specifically designed for this task based on GNN. The evaluation metrics include sensitivity and balanced accuracy (the average of sensitivity and true negative rate).

As demonstrated in TABLE IV, NetTAG significantly outperforms ReIGNN in both metrics. It achieves a 44% improvement in sensitivity (90% vs. 46%) for state register identification and a 13% increase in overall accuracy (86% vs. 73%), highlighting its superior capability of capturing structural and functional information from register cones.

Task3: Endpoint register slack prediction. This task focuses on predicting sign-off timing slack at the netlist stage to provide early feedback, thereby expanding the optimization space in the physical design flow. The prediction is highly challenging due to substantial graph topology changes during physical design optimizations, as discussed in [2]. We compare NetTAG with the state-of-the-art GNN-based method from [2], extending their post-placement method to work at the post-synthesis stage. Performance is evaluated using correlation coefficient (R) and mean absolute percentage error (MAPE).

TABLE IV: Performance comparison on Task2: state/data register identification & Task3: endpoint register slack prediction.

Design	Task 2				Task 3			
	REIGNN [15]		NetTAG		GNN*		NetTAG	
	Sens. (%)	Acc. (%)	Sens. (%)	Acc. (%)	R	MAPE (%)	R	MAPE (%)
itc1	50	72	100	98	0.89	13	0.94	9
itc2	100	92	100	100	0.91	10	0.93	9
chipyard1	30	65	80	79	0.72	17	0.7	20
chipyard2	30	65	90	86	0.86	12	0.95	9
vex1	50	74	82	74	0.94	11	0.93	12
vex2	32	60	86	82	0.97	18	0.97	9
opencores1	42	73	93	84	0.99	26	0.92	29
opencores2	37	80	92	82	0.93	30	0.98	26
Avg.	46	73	**90**	**86**	0.9	17	**0.92**	**15**

* We adapt the GNN model from [2] for netlist-stage slack prediction, as it was originally designed for the layout stage.

As shown in TABLE IV, NetTAG outperforms the customized GNN baseline [2], achieving more accurate slack prediction with R of 0.92 vs. 0.9 and a MAPE of 15% vs. 17%.

Task4: Overall circuit power/area prediction. This task predicts final layout power and area metrics at the netlist stage, offering early estimates for physical design. We compare NetTAG with the synthesis tool and a customized GNN model adopted from PowPrediCT [7], a layout-stage state-of-the-art optimization-aware power predictor.

For area prediction, NetTAG achieves a 4% MAPE without optimization and 11% with optimization, outperforming the GNN's 5% and 18% and the synthesis EDA tool's 5% and 34%. For power, it yields an 8% MAPE without optimization and 12% with optimization, surpassing the GNN's 12% and 19% and the EDA tool's 34% and 38%. These results highlight NetTAG's strong accuracy and robustness for circuit-level tasks.

TABLE V: Performance comparison on Task4: overall circuit power/area prediction.

Target Metric†		EDA Tool		GNN*		NetTAG	
		R	MAPE (%)	R	MAPE (%)	R	MAPE (%)
Area	w/o opt	0.99	5	0.99	5	**0.99**	**4**
	w/ opt	0.95	34	0.95	18	**0.96**	**11**
Power	w/o opt	0.99	34	0.99	12	**0.99**	**8**
	w/ opt	0.73	38	0.76	19	**0.86**	**12**

† Post-layout labels are collected in two scenarios: without considering optimization (denoted as "w/o opt") and with optimization as illustrated in [7] (denoted as "w/ opt").
* We adapt the GNN model from [7] for netlist power and area prediction, as it was originally designed for the layout-stage power prediction.

C. Comparision with pre-trained netlist encoders

In addition to the task-specific methods, we also compare NetTAG with pre-trained netlist encoders. Since they are all limited to combinational AIG circuits, we evaluate them only on Task 1, using an AIG-format dataset. As shown in Fig. 5, NetTAG outperforms SOTA AIG-based encoders, including FGNN [29] and DeepGate3 [27], achieving the highest average performance across all metrics. Notably, NetTAG achieves superior accuracy on the AIG dataset compared with more diverse gate datasets in TABLE III for Task 1, underscoring its adaptability. We also evaluate the standalone ExprLLM component of NetTAG, which performs well using symbolic expressions, highlighting the power of gate semantic understanding. These results emphasize NetTAG's advantage in combining semantic and structural information through the TAG format.

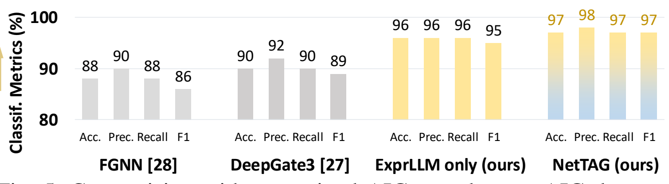


Fig. 5: Comparision with pre-trained AIG encoders on AIG dataset.

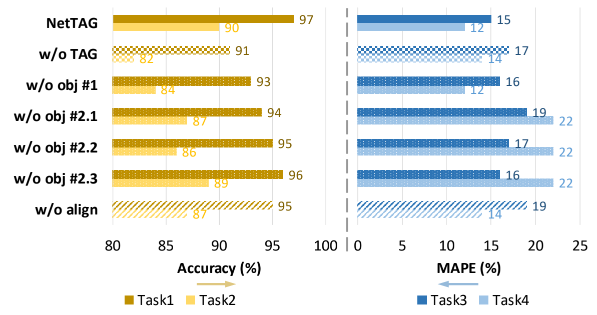


Fig. 6: Ablation study.

D. Ablation Study

This ablation study evaluates the contributions of three key aspects of NetTAG: TAG-based representation, self-supervised pre-training objectives, and cross-stage alignment. The analysis, shown in Fig. 6, highlights the impact of each component on the model's performance across the four tasks, with detailed analysis as follows:

Effectiveness of learning netlist via TAG. Removing text attributes and relying only on graph structure significantly reduces performance, especially on functional tasks. This indicates that semantic information from text attributes is crucial. Additionally, physical tasks also show a slight decline, suggesting that even structure-focused tasks benefit from the added semantic context in TAG.

Effectiveness of self-supervised pre-training objectives. The results demonstrate that the expression contrastive learning objective (#1) for ExprLLM has the greatest impact on functional tasks, suggesting it enhances the LLM's understanding of symbolic logic expressions. Objectives #2.1 (masked gate reconstruction) and #2.2 (graph contrastive learning) improve performance across both functional and physical tasks, highlighting their role in capturing both local and global netlist structure. Objective #2.3 (graph size prediction) shows the strongest effect on physical tasks, where a comprehensive understanding of the netlist's overall structure is essential.

Effectiveness of cross-stage alignment. Removing cross-stage alignment leads to a notable drop in performance across all four tasks. This alignment effectively enhances NetTAG's ability to integrate both early-stage functional and late-stage physical information, which is crucial for downstream tasks.

E. Runtime Analysis

In TABLE VI, we present NetTAG's runtime across various benchmarks. Most runtime is spent on preprocessing (i.e., converting netlists into TAG format) and node-level inference using ExprLLM. Despite these steps, NetTAG demonstrates around a 10x speedup over traditional physical design workflows using commercial EDA tools. There is significant potential for further runtime improvements. For example, symbolic expression extraction for each gate can be highly parallelized, reducing preprocessing time. Additionally, the ExprLLM inference time could be largely reduced by scaling up GPU resources, as this runtime depends on GPU performance and quantity.

IV. Discussion

A. Scalability: Performance Scaling with Model and Data Size

In Fig. 7, we study how the downstream task performance of NetTAG scales with both model size and pre-training data size.

TABLE VI: Runtime (minutes) Comparision.

Source	EDA Tool	Ours (Avg.)			
	P&R (Avg.)	Pre⋆	Infer		Total
			ExprLLM	TAGFormer	
ITC99	164	2	5	0	7
OpenCores	288	18	12	1	31
Chipyard	251	15	10	1	26
VexRiscv	207	8	5	2	15
GNNRE	/	4	2	0	6

⋆ Preprocessing (chunking into cones and converting netlist into TAG).

The plot shows the performance of each task after fine-tuning. Scaling ExprLLM backbone from 110M parameters (BERT) to larger models like Meta Llama 3.1 with 1.3B and 8B parameters results in significant performance improvements across all four tasks. Similarly, expanding data size from 25% to 100% of the dataset consistently enhances performance. These results demonstrate the scalability of NetTAG, suggesting that further increases in model and data size could lead to even greater improvements in accuracy and generalization.

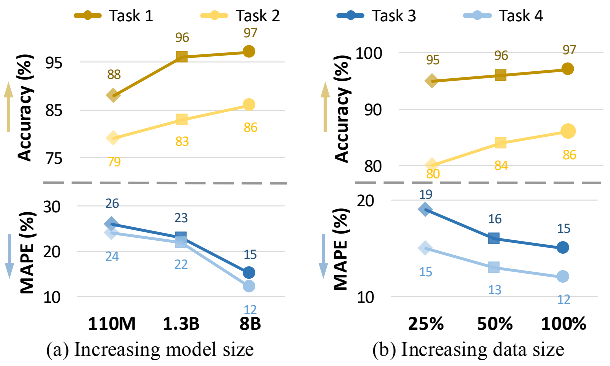
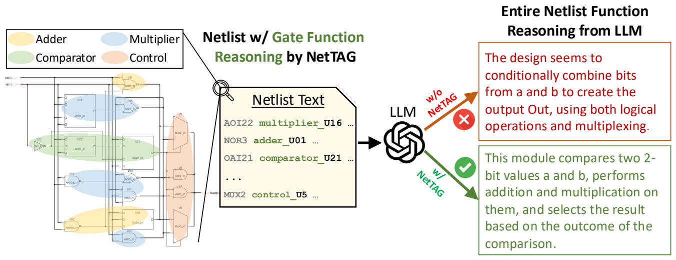


Fig. 7: Performance scaling with model and data size.

B. Demo: Reasoning Netlist's Arithmetic Function with NetTAG

Fig. 8 shows how NetTAG improves functional reasoning for arithmetic netlists. Using an LLM (e.g., OpenAI's o1-preview) to analyze post-synthesis netlist Verilog texts, we assess its ability to interpret the arithmetic function from the flattened netlists. Without NetTAG's gate-level function identification, the LLM struggles with interpreting functionality from flattened netlists. When integrated with NetTAG, the LLM accurately infers that the netlist module compares two 2-bit values, performs addition and multiplication, and selects the result based on the outcome of the comparison.


Adder
Multiplier
Comparator
Control
Netlist w/ Gate Function Reasoning by NetTAG
Entire Netlist Function Reasoning from LLM
Netlist Text
LLM
w/o NetTAG
w/ NetTAG
```
AOI22 multiplier_U16 …
NOR3 adder_U01 …
OAI21 comparator_U21 …
...
MUX2 control_U5 …
```

The design seems to conditionally combine bits from a and b to create the output Out, using both logical operations and multiplexing.

This module compares two 2-bit values a and b, performs addition and multiplication on them, and selects the result based on the outcome of the comparison.

Fig. 8: A demo for reasoning arithmetic function from netlists.

V. Conclusion And Future Work

In this paper, we present NetTAG, a foundation model for netlist representation learning that accommodates diverse gate types and supports a range of functional and physical tasks. By formulating netlists as text-attributed graphs, NetTAG uniquely integrates gate semantics with graph structure through its multimodal architecture and self-supervised pre-training objectives. Future work will enhance NetTAG by extending pre-training to larger circuit datasets and exploring decoding methods to broaden task support capabilities.

VI. Acknowledgement

This work is supported by Hong Kong Research Grants Council (RGC) CRF Grant C6003-24Y, National Natural Science Foundation of China (NSFC) 62304192, and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK, Hong Kong SAR. We thank HKUST Fok Ying Tung Research Institute and National Supercomputing Center in Guangzhou Nansha Sub-center for computational resources.

References

[1] W. Fang, S. Liu, H. Zhang, and Z. Xie, "Annotating slack directly on your verilog: Fine-grained rtl timing evaluation for early optimization," in *Proceedings of 2024 ACM/IEEE Design Automation Conference (DAC)*. ACM, 2024, pp. 1–6.

[2] Z. Wang, S. Liu *et al.*, "Restructure-tolerant timing prediction via multimodal fusion," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[3] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin, "A timing engine inspired graph neural network model for pre-routing slack prediction," in *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 1207–1212.

[4] W. Fang, Y. Lu, S. Liu, Q. Zhang, C. Xu, L. W. Wills, H. Zhang, and Z. Xie, "Masterrtl: A pre-synthesis ppa estimation framework for any rtl design," in *Proceedings of 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[5] C. Xu, P. Sharma, T. Wang, and L. W. Wills, "Fast, robust and transferable prediction for hardware logic synthesis," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 167–179.

[6] P. Sengupta, A. Tyagi, Y. Chen, and J. Hu, "How good is your verilog rtl code? a quick answer from machine learning," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2022, pp. 1–9.

[7] Y. Du, Z. Guo, X. Jiang, Z. Chai, Y. Zhao, Y. Lin, R. Wang, and R. Huang, "Powpredict: Cross-stage power prediction with circuit-transformation-aware learning," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*. ACM, 2024, pp. 1–6.

[8] Z. Xie, X. Xu, M. Walker, J. Knebel, K. Palaniswamy, N. Hebert, J. Hu, H. Yang, Y. Chen, and S. Das, "APOLLO: An automated power modeling framework for runtime power introspection in high-volume commercial microprocessors," in *Proceedings of 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021.

[9] Y. Zhou, H. Ren, Y. Zhang, B. Keller, B. Khailany, and Z. Zhang, "PRIMAL: Power inference using machine learning," in *Proceedings of 56th Annual Design Automation Conference (DAC)*, 2019.

[10] Z. Xie, Y.-H. Huang *et al.*, "Routenet: Routability prediction for mixed-size designs using convolutional neural network," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[11] S. Liu, Q. Sun, P. Liao, Y. Lin, and B. Yu, "Global placement with deep learning-enabled explicit routability optimization," in *Proceedings of 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1821–1824.

[12] S. Zheng, L. Zou, P. Xu, S. Liu, B. Yu, and M. Wong, "Lay-net: Grafting netlist knowledge on layout-based congestion prediction," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[13] N. Wu, Y. Li, C. Hao, S. Dai, C. Yu, and Y. Xie, "Gamora: Graph learning based symbolic reasoning for large-scale boolean networks," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[14] L. Alrahis, A. Sengupta, J. Knechtel, S. Patnaik, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu, "Gnn-re: Graph neural networks for reverse engineering of gate-level netlists," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 8, pp. 2435–2448, 2021.

[15] S. D. Chowdhury, K. Yang, and P. Nuzzo, "Reignn: State register identification using graph neural networks for circuit reverse engineering," in *Proceedings of IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[16] Z. He, Z. Wang *et al.*, "Graph learning-based arithmetic block identification," in *Proceedings of IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–8.

[17] L. Chen, Y. Chen *et al.*, "Large circuit models: opportunities and challenges," *Science China Information Sciences*, 2024.

[18] W. Fang, J. Wang, Y. Lu, S. Liu, Y. Wu, Y. Ma, and Z. Xie, "A survey of circuit foundation model: Foundation ai models for vlsi circuit design and eda," *arXiv preprint arXiv:2504.03711*, 2025.

[19] W. Fang, S. Liu, J. Wang, and Z. Xie, "Circuitfusion: multimodal circuit representation learning for agile chip design," in *International Conference on Learning Representations (ICLR)*, 2025.

[20] K. Zhu, H. Chen, W. J. Turner, G. F. Kokai, P.-H. Wei, D. Z. Pan, and H. Ren, "Tag: Learning circuit spatial embedding from layouts," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2022, pp. 1–9.

[21] A. Sohrabizadeh, Y. Bai, Y. Sun, and J. Cong, "Robust gnn-based representation learning for hls," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023.

[22] Z. Luo, T. S. Hy *et al.*, "De-hnn: An effective neural model for circuit netlist representation," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2024.

[23] C. Deng, Z. Yue *et al.*, "Less is more: Hop-wise graph attention for scalable and generalizable learning on circuits," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*. ACM, 2024.

[24] S. Yang, Z. Yang, D. Li, Y. Zhang, Z. Zhang, G. Song, and J. Hao, "Versatile multi-stage graph neural network for circuit representation," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 20 313–20 324, 2022.

[25] M. Li, S. Khan, Z. Shi, N. Wang, H. Yu, and Q. Xu, "Deepgate: Learning neural representations of logic gates," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 667–672.

[26] Z. Shi, H. Pan *et al.*, "Deepgate2: Functionality-aware circuit representation learning," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[27] Z. Shi, Z. Zheng, S. Khan, J. Zhong, M. Li, and Q. Xu, "Deepgate3: towards scalable circuit representation learning," *arXiv preprint arXiv:2407.11095*, 2024.

[28] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 61–66.

[29] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, Y. Huang, and B. 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 (TCAD)*, 2024.

[30] S. Khan, Z. Shi, M. Li, and Q. Xu, "Deepseq: Deep sequential circuit learning," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–2.

[31] W. Fang, S. Liu, H. Zhang, and Z. Xie, "A self-supervised, pre-trained, and cross-stage-aligned circuit encoder provides a foundation for various design tasks," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2025.

[32] Z. Pei, H. Zhen, M. Yuan, Y. Huang, and B. Yu, "Betterv: Controlled verilog generation with discriminative guidance," in *Forty-first International Conference on Machine Learning (ICML)*, 2024.

[33] W. Fang, M. Li, M. Li, Z. Yan, S. Liu, H. Zhang, and Z. Xie, "Assertllm: Generating and evaluating hardware verification assertions from design specifications via multi-llms," *arXiv preprint arXiv:2402.00386*, 2024.

[34] H. Wu, Z. He *et al.*, "Chateda: A large language model powered autonomous agent for eda," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[35] X. He, X. Bresson, T. Laurent, A. Perold, Y. LeCun, and B. Hooi, "Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[36] H. Liu, J. Feng *et al.*, "One for all: Towards training one graph model for all classification tasks," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[37] M. Gario and A. Micheli, "Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms," in *SMT workshop*, 2015.

[38] P. BehnamGhader, V. Adlakha, M. Mosbach, D. Bahdanau, N. Chapados, and S. Reddy, "Llm2vec: Large language models are secretly powerful text encoders," *arXiv preprint arXiv:2404.05961*, 2024.

[39] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "Mteb: Massive text embedding benchmark," in *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, 2023, pp. 2014–2037.

[40] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[41] F. Corno, M. S. Reorda, and G. Squillero, "Rt-level itc'99 benchmarks and first atpg results," *IEEE Design & Test of computers (ITC)*, 2000.

[42] *OpenCores: The reference community for Free and Open Source gateware IP cores*, https://opencores.org/.

[43] A. Amid, D. Biancolin *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, 2020.

[44] VexRiscv, "VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation," 2022. [Online]. Available: https://github.com/SpinalHDL/VexRiscv

[45] A. Dubey, A. Jauhri *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[46] Q. Wu, W. Zhao *et al.*, "Simplifying and empowering transformers for large-graph representations," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2024.

[47] C. Lee, R. Roy *et al.*, "Nv-embed: Improved techniques for training llms as generalist embedding models," *arXiv preprint arXiv:2405.17428*, 2024.