A Self-Supervised, Pre-Trained, and Cross-Stage-Aligned Circuit Encoder Provides a Foundation for Various Design Tasks

Wenji Fang
HKUST
wfang838@connect.ust.hk

Shang Liu
HKUST
sliudx@connect.ust.hk

Hongce Zhang
HKUST & HKUST(GZ)
hongcezh@ust.hk

Zhiyao Xie*
HKUST
eezhiyao@ust.hk

ABSTRACT

Machine learning (ML) techniques have shown remarkable effectiveness in electronic design automation (EDA). Traditionally, most ML for EDA approaches are task-specific, requiring a tedious development process of a tailored ML model for each individual design task. Recently, circuit representation learning has emerged as a promising trend. This approach converts circuits into embeddings, which can then be adapted to distinct downstream tasks. However, existing methods still fall short of providing a truly general circuit representation that supports highly diverse tasks. In this work, we introduce *CircuitEncoder*, a self-supervised, pre-trained, and cross-stage-aligned general circuit encoder. It provides a general foundation for diverse ML-based EDA tasks, including both design quality and functional reasoning. CircuitEncoder is pre-trained through multi-stage contrastive learning utilizing unlabeled circuits. It encodes circuits from different design stages into embedding vectors within shared latent space, facilitating fine-tuning for various downstream tasks. CircuitEncoder outperforms the state-of-the-art task-specific supervised solutions for multiple EDA tasks, including design quality tasks for register-transfer level (RTL)-stage timing and area prediction, as well as functional tasks for netlist-stage state register identification.

ACM Reference Format:
Wenji Fang, Shang Liu, Hongce Zhang, and Zhiyao Xie. 2025. A Self-Supervised, Pre-Trained, and Cross-Stage-Aligned Circuit Encoder Provides a Foundation for Various Design Tasks. In *30th Asia and South Pacific Design Automation Conference (ASPDAC '25), January 20–23, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3658617.3697597

1 INTRODUCTION

Machine learning (ML) techniques have demonstrated remarkable effectiveness in electronic design automation (EDA). Existing ML for EDA works are mostly task-specific, tailoring a dedicated ML model for each specific EDA task individually. They may target either the prediction of design objectives (e.g, timing [9, 13, 29], area [10, 11, 20, 22, 35], power [33, 34, 40], and routability [18, 32, 39]) or the reasoning of circuit functionalities [5, 15, 31]. They are mostly developed by supervised training on a carefully crafted dataset for the

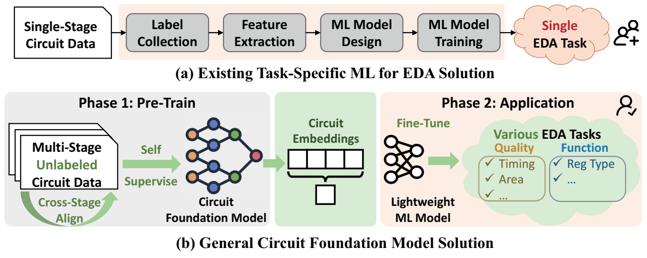


Figure 1: The target ML for EDA paradigm based on CircuitEncoder. The self-supervised, pre-trained, cross-stage-aligned CircuitEncoder encodes circuit components into embeddings with rich circuit information, based on which lightweight ML solutions are easily developed for various design tasks.

specific task. Despite obvious effectiveness, there are two limitations in this mainstream supervised ML for EDA paradigm. 1) The development process of supervised task-specific solutions is tedious and time-consuming. The development steps include circuit collection, label generation, feature engineering, model selection, model training and testing. This whole process easily takes months of engineering efforts. 2) Since supervised task-specific models cannot be generalized to other tasks, it leads to an inefficient and repetitive development of ML solutions. More importantly, it implies that these ML solutions only capture task-specific patterns, instead of more general and complete information from target circuit components.

Recently, circuit representation learning [4, 16, 17, 23, 25, 28, 36] was explored to separate the ML for EDA process into two separate phases. In phase 1, circuit models will learn to generate a general representation (i.e., an embedding vector) for each circuit component. In phase 2, downstream ML models can be easily trained for each downstream task, directly taking the representation instead of a complex raw circuit as model input. However, we argue that existing works still do not provide perfectly general circuit representation. For example, existing works only support one type of design task, either design quality prediction [16, 36] or functional reasoning [17, 23, 25, 28]. No prior work can provide a *truly general representation* that supports largely different design tasks. In addition, prior works only focus on one specific design stage, such as RTL or netlist, without *alignment* across different stages.

In this work, we propose a more general circuit representation learning method named *CircuitEncoder*, which targets as a *general foundation* for various ML-based EDA tasks. A comparison between CircuitEncoder and existing works is given in Table 1, and our target paradigm is summarized in Figure 1. In phase 1, CircuitEncoder is pre-trained on unlabeled circuits by us and released to the public[1].

*Corresponding Author.

[1]The code and pre-trained CircuitEncoder model are available at https://github.com/hkust-zhiyao/CircuitEncoder. We will maintain this general CircuitEncoder model and further pre-train it on more unlabeled circuit samples.

Table 1: Existing two-phase circuit representation learning techniques for ASIC design.

Method	Downstream Tasks			Pre-Training		Design Stage		Support Seq. Circuit	Open-Source Model
	Multi-Type	Design Quality	Function	Self-Supervised	Train Task	Cross-Stage	Target Stage		
Design2Vec [25]			✓		Cover Point		RTL	✓	
SNS v2 [36]		✓		✓	Contrastive		RTL	✓	
FGNN [28]			✓	✓	Contrastive		Netlist		
DeepGate [17]			✓		Probability		Netlist		
DeepGate2 [23]			✓		Truth Table		Netlist		✓
DeepSeq [16]		✓✗*			Probability		Netlist	✓	
CircuitEncoder (Ours)	✓	✓	✓	✓	**Multi-Stage Contrastive**	✓	**RTL Netlist**	✓	✓

* DeepSeq predicts netlist gate toggle rate at the node level to estimate power consumption, rather than directly modeling power.

It can convert the given circuit component to a general embedding, encoding rich circuit information in it. This circuit embedding is the direct model input for different types of design tasks, including both design quality prediction and functional reasoning. In phase 2, users can fine-tune lightweight downstream models based on embeddings. The few-shot transfer solution proves to significantly outperform supervised solutions based on raw circuits. Similar to the benefit of foundation models like BERT [8] or CLIP [21] in natural language processing and computer vision, open-sourced pre-trained CircuitEncoder will be widely accessible and support the agile development of better ML solutions for various design tasks.

The multi-task generalization ability of CircuitEncoder implies that it captures rich circuit information. This is achieved based on the following two innovative learning policies: ① CircuitEncoder incorporates multi-stage self-supervised contrastive learning in its learning process. Through contrastive learning within and across the design stages, the encoder learns to maximize the similarity in the embeddings of the sub-circuits with the same functionality (i.e., positive samples), and minimize the similarity for negative samples. ② CircuitEncoder *aligns* circuit embedding at different design stages within a shared latent space. In a typical design flow, the earlier design stage (i.e., higher abstraction level) captures more semantic content, while the later design stage (i.e., lower abstraction level) provides complex implementation details. The cross-stage alignment benefits both predictions of downstream design qualities at early stages and reasoning of circuit functionality at late stages. We summarize the contributions of CircuitEncoder as below:

- To the best of our knowledge, CircuitEncoder is the first general method to achieve high performance in largely distinct tasks, including design quality tasks for RTL-stage timing and area modeling, as well as functional reasoning tasks for netlist-stage state register identification. This indicates the CircuitEncoder's great generalization ability and potential to provide a "foundation" for various tasks.
- For pre-training (phase 1), we propose a multi-stage contrastive learning scheme, training within and across design stages at the sub-circuit level. Based on this, we align circuit embeddings from different stages within a shared latent space, capturing rich circuit information.
- For application to various tasks (phase 2), the pre-trained CircuitEncoder generates embeddings for each new sub-circuit. These embeddings serve as the general inputs to fine-tune for various downstream tasks, avoiding tedious task-specific feature engineering and model design.
- According to experiments, the general two-phase paradigm supported by CircuitEncoder significantly outperforms task-specific supervised solutions. It shows a 5% and 4% lower MAPE for timing and area evaluations, respectively, and achieves a 15% higher accuracy for functional identification.

2 PRELIMINARIES AND PROBLEM FORMULATION

2.1 Circuit Representation Learning

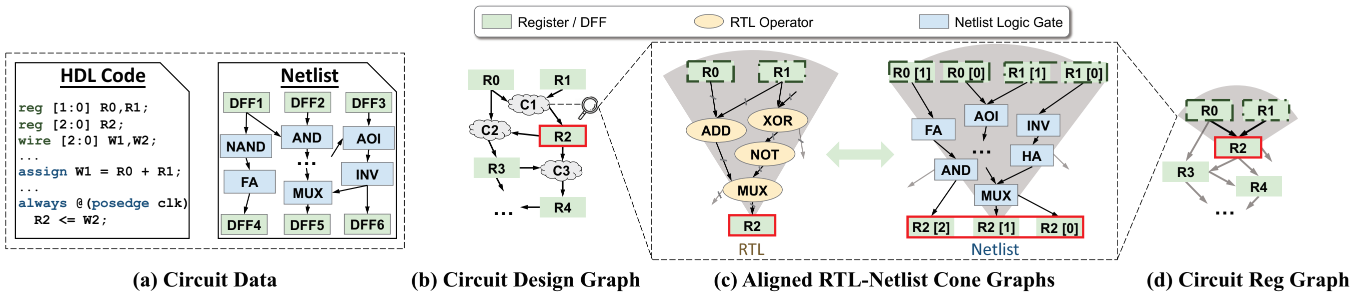
Table 1 summarizes the existing circuit representation learning approaches for ASIC design. Most methods focus on functional reasoning, while others target design quality evaluation. Specifically, Design2Vec [25] and the DeepGate family [17, 23] leverage functional supervisions like cover points, signal probability, and truth tables for pre-training, subsequently applying these models to functional tasks such as testing, logic equivalence checking, and SAT solving. FGNN [28] uses contrastive pre-training to identify gate functionality. For design quality, SNS v2 [36] employs contrastive pre-training and predicts timing, power, and area metrics. DeepSeq [16] models the sequential behavior of circuits, predicting gate toggle rates, which are the input to power simulation tools.

However, these models are not general enough to support both design quality and functional reasoning tasks. Many works [17, 23, 28] only apply to small combinational components. Additionally, These methods focus on specific design stages, either the RTL stage [25, 36] or the netlist stage [16, 17, 23, 28], lacking information learned from earlier or later stages.

Besides the two-phase strategy, several works [7, 12, 37] propose customized circuit learning models supporting supervised training for various downstream EDA tasks. Additionally, efforts have been made to develop representation learning methods tailored for other types of circuit design, such as high-level synthesis for FPGAs [24] and analog and mixed-signal circuits [41].

2.2 Problem Formulation

Our target is to develop a CircuitEncoder (f_{CE}) that can generate embeddings E_R and E_N for any RTL and netlist design, providing a foundation for various downstream tasks, including design quality tasks (T_Q) and functional reasoning tasks (T_F). We denote the RTL design as $\mathcal{R}$ and the gate-level netlist as $\mathcal{N}$. CircuitEncoder is pre-trained on unlabeled circuits across design stages to learn



```
reg [1:0] R0,R1;
reg [2:0] R2;
wire [2:0] W1,W2;
...
assign W1 = R0 + R1;
...
always @(posedge clk)
  R2 <= W2;
```

Figure 2: Circuit data preprocessing and cross-stage alignment. (a) RTL code and gate-level netlist circuits. (b) Circuit in graph format. (c) Aligned circuit cone graphs. RTL and netlist graphs are aligned based on registers. Each register backtracks all combinational logic until reaching all other registers, ensuring functionally equivalent cross-stage alignment at the sub-circuit level. (d) Circuit register dependency graph. All register cones from the same design are aggregated for the circuit-level tasks.

cross-stage circuit information, and then it can be easily applied to various design tasks:

CircuitEncoder Pre-Training. During pre-training, CircuitEncoder encodes the aligned RTL and netlist and learns cross-stage embeddings in a shared latent space through self-supervised tasks:

$$\text{pre-train}(f_{CE}(\mathcal{R}\ \&\ \mathcal{N})) \rightarrow \{E_R\ \&\ E_N\}. \tag{1}$$

CircuitEncoder Application. Given a new circuit, CircuitEncoder generates embeddings supporting few-shot inferences for downstream tasks. The downstream models (f_D) are fine-tuned based on the embeddings and specific task labels. This application process is formulated below:

$$\begin{aligned} &f_{CE}(\mathcal{R}) = E_R,\ f_{CE}(\mathcal{N}) = E_N, \\ &\text{fine-tune: } \{f_D(E_R),\ f_D(E_N)\} \rightarrow \{T_Q, T_F\}. \end{aligned} \tag{2}$$

3 CIRCUIT ENCODER FRAMEWORK

Figure 1 (b) illustrates the overall CircuitEncoder workflow. Circuits are encoded into embeddings that cluster functionally similar circuits more closely. We propose to align the RTL and netlist design stages to bridge the gap between logic and physical design processes. As detailed in this section, the model is pre-trained to learn the cross-stage-aligned representations. Subsequently, the pre-trained CircuitEncoder is applied to various downstream tasks.

3.1 Circuit Data Preprocessing and Alignment

3.1.1 Circuit-to-Graph Transformation. The original circuit data, as shown in Figure 2 (a), consists of RTL code and gate-level netlists, which cannot be directly processed by ML models. Here, we initially convert circuits at both stages into the graph format. First, for RTL data in HDL code format, we convert the code into an abstract syntax tree (AST) and subsequently extract the graph based on this tree, a process similar to that described in [10]. The nodes of the RTL graph include word-level register signals and operators (e.g., And, Add, Equal, Mux), with the wires in HDL code forming the edges connecting these elements. Second, for gate-level netlists, the transformation is more straightforward: flip-flops (FF) and logic gates (e.g., AOI, INV, FA, AND) serve as nodes, and the wires connecting these gates form the edges of the graph. The graph format of circuits is demonstrated in Figure 2 (b).

3.1.2 Sub-Circuit Format with RTL-Netlist Alignment. Figure 2 (c) illustrates the alignment for RTL and netlist graphs at the sub-circuit level. Sequential registers (including primary input/output pins) remain unchanged[2] during the logic synthesis and physical design processes, allowing for the one-to-one mapping between register signals in RTL and flip-flops in the netlist. This mapping forms the foundation for cross-stage alignment across distinct abstraction layers.

We propose to extract sub-circuits from both RTL and netlist graphs based on the mapped registers. Specifically, for each mapped register pair (i.e., word-level register signal in RTL and the corresponding multi-bit flip-flops in netlist), we backtrack all the combinational logics in both graphs until reaching all other registers. These specific sub-circuits are termed the ***cone*** of the register.

We summarize the advantages of the cone format: (1) The extracted register cones between RTL and netlist are strictly aligned and functionally equivalent, ensuring consistency across design stages. (2) Each register cone captures the complete state transition of this register within a single clock cycle. It includes the complete timing paths and logic computations for the register. This enables learning the circuit's sequential behavior, which is essential for tasks such as timing evaluation and sequential functional reasoning. (3) The cone format serves as an intermediate granularity level. It bridges the gap between fine-grained nodes and coarse-grained overall design graphs, enhancing the model's adaptability for sub-circuit level tasks.

3.2 Self-Supervised Pre-Training Task

To learn the general intrinsic circuit information, we propose to apply self-supervised learning [19] to unlabeled circuit data. It enables CircuitEncoder to learn intrinsic semantics and structure of numerous circuits through well-designed pretext tasks, without supervision labels to guide learning for specific circuit properties. This significantly enhances the model's ability to generalize across various tasks and datasets.

In our work, we propose multi-stage contrastive learning to learn circuits within and across distinct design stages, as illustrated in Figure 3. Contrastive learning is utilized to cluster functionally similar circuits closely together, while others are distanced from each other. We incorporate two contrastive schemes: intra-stage

[2]Although there are optimizations for registers such as retiming, pipelining, and ungrouping, most registers remain unchanged across the design stages.

(depicted in orange and blue) and inter-stage (shown in green) contrastive learning, each with distinct loss functions.

Functionally Equivalent Circuit Augmentation. Since circuits with the same functionality contain diverse structures, we propose to augment both RTL and netlist data using functionally equivalent circuit transformations. These transformations maintain the original semantics but change the structure of the circuit graph, effectively serving as positive sample augmentation, denoted as RTL$^+$ and Net$^+$ in Figure 3. This method follows similar processes as those detailed in [28, 36]. For negative samples, we randomly select non-equivalent circuit cones, shown as RTL$^-$ and Net$^-$. This circuit augmentation approach effectively facilitates contrastive learning, which differentiates between functionally equivalent and non-equivalent circuits.

❶ **Intra-Stage Contrastive Learning.** Our intra-stage contrastive learning approach focuses on individually learning RTL and netlist circuits within their respective design stages. For the RTL stage, the goal is to preserve similarity within equivalent RTL cones and to differentiate between non-equivalent cones. CircuitEncoder encodes the original, positively augmented, and negatively augmented RTL cone graphs into cone-level embedding vectors E_{R}, E_{R^+} and E_{R^-}. The contrastive learning is then employed to minimize the distance between E_{R} and E_{R^+} while maximizing the distance between E_{R} and E_{R^-}. The same methodology is applied to netlist cones to ensure consistent and effective contrastive learning within each design stage. The loss functions for the inter-stage contrastive learning are defined below:

$$\begin{aligned} L_{rr} &= max(\left\|E_{\text{R}} - E_{\text{R}^+}\right\|_2 - \|E_{\text{R}} - E_{\text{R}^-}\|_2 + m_{rr}, 0), \\ L_{nn} &= max(\left\|E_{\text{N}} - E_{\text{N}^+}\right\|_2 - \|E_{\text{N}} - E_{\text{N}^-}\|_2 + m_{nn}, 0), \end{aligned} \tag{3}$$

where m_{rr} and m_{nn} are the margins to balance the positive and negative similarities. This loss function with max applies a penalty only when the margin between the similarities of positive and negative pairs is not met, focusing learning efforts on more challenging and incorrectly classified samples.

❷ **Inter-Stage Contrastive Learning.** Alignment is crucial to enhance the CircuitEncoder's ability to accurately interpret and predict across distinct design stages. Most prediction tasks need to infer the characteristics of an unknown circuit at an earlier or later stage. Utilizing the cross-stage alignment insights, CircuitEncoder effectively enhances design quality predictions at earlier stages and functional reasoning at later stages.

To facilitate a bidirectional understanding between high-level RTL semantics and low-level netlist implementations, we introduce an inter-stage contrastive learning scheme to capture the cross-stage interactions. Initially, CircuitEncoder encodes the aligned RTL and netlist cone graphs into embeddings. Subsequently, CircuitEncoder is trained using a cross-stage contrastive loss to align the embeddings within a shared latent space. This enables CircuitEncoder to express high-level RTL with low-level netlist implementations and, meanwhile, understand the low-level netlist with high-level RTL semantics. The cross-stage contrastive loss function is detailed as follows:

$$\begin{aligned} L_{rn} = &\, max(\left\|E_{\text{R}} - E_{\text{N}^+}\right\|_2 - \|E_{\text{R}} - E_{\text{N}^-}\|_2 + m_{rn}, 0) \\ &+ max(\left\|E_{\text{N}} - E_{\text{R}^+}\right\|_2 - \|E_{\text{N}} - E_{\text{R}^-}\|_2 + m_{rn}, 0). \end{aligned} \tag{4}$$

Based on the two contrastive learning schemes, we formulate the complete loss for our multi-stage contrastive learning as follows:

$$L_{CL} = \alpha_{rr}L_{rr} + \alpha_{nn}L_{nn} + \alpha_{rn}L_{rn}, \tag{5}$$

where α_{rr}, α_{nn}, and α_{rn} are trade-off hyperparameters among different contrative losses.

3.3 Encoder Model and Feature Extraction

In this subsection, we illustrate the CircuitEncoder's ML models and extracted features for RTL and netlist designs. The detailed model architecture and hyperparameters will be provided later in Section 4.1.

3.3.1 *Graph Transformer for RTL-Stage.*

To encode the RTL design into embedding vectors, we apply a graph transformer model to effectively capture the complex relationships of the semantics and structure of RTL design. The transformer's self-attention and positional encoding mechanisms allow it to consider the entire graph context, facilitating the generation of rich, context-aware embeddings at both the node and cone levels for the RTL design. The graph transformer demonstrates significant enhancement compared to traditional GNN models, as detailed further in Section 4.4.

Initially, we extract features for each node and edge within the RTL cone graph. Node features include bit-width and operator type, represented in one-hot encoding. Edge features are determined by the types of nodes they connect, and also encoded in one-hot format.

Leveraging these features, we employ three graph positional encodings, similar to our backbone [38], to effectively capture the information of the circuit graph: (1) Centrality encoding enriches each node's features with numbers of fan-in and fan-out. (2) Spatial encoding computes the shortest path distances between all connected node pairs, applying them as a bias term in the self-attention module. (3) Edge encoding utilizes the edge features along one of the shortest paths between node pairs, also contributing as a bias term in the self-attention module.

Please note that there is a special global node in the graph transformer, denoted as `[C]` shown in Figure 3. It is added to the original graph and connected with each node, forming individual edges. After training, due to the self-attention mechanism, the embedding of `[C]` serves as the sub-circuit level embedding for the register cone.

3.3.2 *Graph Neural Network for Netlist-Stage.*

Although the graph transformer's capability is powerful, it's resource-consuming for large-scale graphs, such as netlist graphs, which are significantly larger than RTL graphs. Therefore, we use a GNN model to encode the netlist cones.

In the netlist graph, the node features include the gate type in one-hot encoding, the number of fan-in and fan-out nodes, and physical characteristics such as driving strength, cell area, output capacitance, and resistance. The GNN aggregates node information through the message-passing mechanism, generating robust node-level embeddings. Subsequently, average graph pooling is applied to aggregate these node embeddings into a cone-level embedding, also denoted as `[C]`.

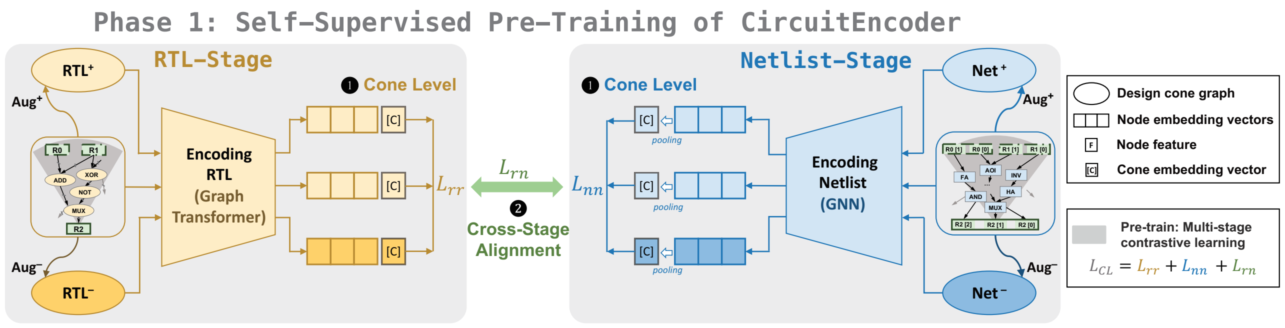


Figure 3: CircuitEncoder cross-stage-aligned self-supervised pre-training workflow. We employ multi-stage contrastive learning to understand circuits within and across design stages at the sub-circuit cone level.

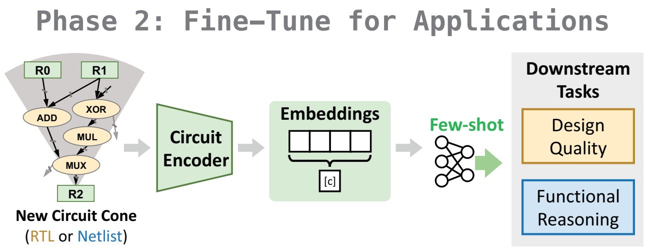


Figure 4: Few-shot fine-tuning for downstream tasks based on CircuitEncoder.

3.4 Fine-Tuning for Downstream Tasks

To apply our CircuitEncoder on distinct downstream tasks, we fine-tune additional task-specific ML models based on task labels, similar to the widely adopted supervised learning. However, our method differs from the supervised methods that process raw circuit data through tedious feature engineering and complex ML models. Instead, we begin with embedding vectors generated by CircuitEncoder. Our task-specific models are lightweight and easily integrated, such as multi-layer perceptrons (MLP), tree-based models, and additional GNN layers.

This two-phase method significantly simplifies ML model development and enhances accuracy across various tasks. Compared to supervised methods, the pre-trained model improves accuracy by capturing rich structural, semantic, and cross-stage information. Additionally, unlike supervised methods that require an extensive labeling process, few-shot fine-tuning with CircuitEncoder uses fewer labeled datasets and still achieves higher accuracy, making it more efficient. The detailed experimental evaluation will be presented in Section 4.

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

Our CircuitEncoder is implemented using Pytorch and DGL [27] for self-supervised pre-training and graph modeling. Experiments are conducted on a server with a 2.9 GHz Intel Xeon(R) Platinum 8375C CPU, 512 GB RAM, and an NVIDIA A4000 GPU.

Table 2 lists the open-source design benchmarks collected for our dataset. We created a dataset of 41 open-source RTL designs, synthesized into gate-level netlists using Synopsys Design Compiler® and the NanGate 45nm technology library, with design metrics obtained from Synopsys Prime Time®. The functionally equivalent RTL code and netlists are augmented using Yosys [30] and ABC [3], then converted into graphs using the similar methods described in [10, 28]. The dataset includes 7,166 RTL and netlist cone pairs, each with positive and negative samples, totaling 42,996 graphs. We employ an 80/20 training/test split based on complete designs, ensuring that no sub-circuits from the same design appear in both training and testing sets. This involves using 33 designs for training and 8 for testing.

Table 2: Benchmark design information.

Source Benchmarks	# Design	Design Size {Min, Avg, Max} #K Gates	# Cones	Original HDL Type
ITC [6]	7	{7, 15, 22}	{12, 21, 31}	VHDL
OpenCores [1]	5	{2, 40, 59}	{12, 96, 173}	Verilog
Vex [26]	17	{8, 208, 591}	{39, 168, 694}	SpinalHDL
Chipyard [2]	12	{11, 49, 194}	{28, 461, 2730}	Chisel

Table 4 summarizes the hyperparameters for the ML models employed in the two-phase processes. For RTL graphs, we utilize Graphormer [38] as our graph transformer backbone, equipped with positional encodings that handle up to 256 fan-ins and fan-outs for centrality encoding, a maximum distance of 5 for spatial encoding, and an edge dimension of 12 for edge encoding. For netlist graphs, we employ GraphSage due to its efficiency in handling larger graphs under computational constraints compared with Graphormer. During contrastive pre-training, we employ the triplet loss function [14] with margin parameters set to 1.0. We maintain the intra-stage weights α_{rr} and α_{nn} at 1.0, and increase the inter-stage weight α_{rn} from 0.4 to 1.2. The Pre-training process lasted 50 epochs (20 hours), followed by fine-tuning with additional lightweight models for only 0.05 hours.

4.2 Downstream Tasks and Baseline Methods

The pre-trained CircuitEncoder model can be easily adapted for both design quality tasks and functional tasks for RTL and netlist designs. All downstream tasks are set up strictly following the baseline methods. We detail these tasks and state-of-the-art (SOTA) baseline methods with corresponding evaluation metrics below:

Design Quality Evaluation. CircuitEncoder is evaluated at two granularities for RTL-stage design quality evaluation tasks:

(1) Register slack prediction for RTL designs *at the cone level*, compared with RTL-Timer [9]. Accurately predicting fine-grained

Table 3: Accuracy comparison for the cone-level tasks for RTL and netlist designs.

	RTL-Stage (Register Slack Prediction)										Netlist-Stage (State Register Identification)									
Method	**RTL-Timer (supervised learning)**						**CircuitEncoder (pre-train + few-shot)**				**ReIGNN⋆ (supervised learning)**						**CircuitEncoder (pre-train + few-shot)**			
% of train	13%		50%		100%		13%		50%		13%		50%		100%		13%		50%	
Test Designs	R	MAPE	R	MAPE	R	MAPE	R	MAPE	R	MAPE	Sens.	Acc.	Sens.	Acc.	Sens.	Acc.	Sens.	Acc.	Sens.	Acc.
ITC1	0.48	22%	0.77	20%	0.82	18%	0.91	21%	0.96	9%	0%	72%	50%	72%	50%	72%	100%	98%	100%	98%
ITC2	0.43	26%	0.83	12%	0.88	10%	0.92	19%	0.96	9%	0%	92%	100%	92%	100%	92%	100%	100%	100%	100%
Chipyard1	0.57	30%	0.89	12%	0.92	18%	0.81	15%	0.83	18%	0%	50%	0%	50%	30%	65%	78%	77%	79%	79%
Chipyard2	0.56	31%	0.85	19%	0.88	12%	0.84	12%	0.85	13%	0%	50%	0%	50%	30%	65%	84%	78%	89%	85%
Vex1	0.28	27%	0.65	15%	0.87	24%	0.69	25%	0.88	26%	0%	50%	0%	50%	50%	74%	76%	79%	82%	72%
Vex2	0.73	29%	0.93	17%	0.86	16%	0.85	13%	0.87	13%	15%	57%	21%	57%	32%	60%	73%	76%	79%	78%
Vex3	0.27	36%	0.56	40%	0.84	16%	0.81	14%	0.89	12%	16%	48%	0%	48%	50%	72%	81%	82%	85%	84%
Vex4	0.12	40%	0.76	18%	0.87	12%	0.83	16%	0.86	14%	30%	63%	33%	63%	33%	63%	88%	79%	90%	81%
Avg.	**0.43**	**30%**	**0.78**	**19%**	**0.87**	**16%**	**0.83**	**17%**	**0.89**	**14%**	**8%**	**60%**	**26%**	**60%**	**47%**	**70%**	**85%**	**84%**	**88%**	**85%**

⋆ ReIGNN further incorporates the structural analysis of the netlist graph to improve ML model predictions. For a fair comparison, we only evaluate the ML model component of ReIGNN against our method. Please note that CircuitEncoder can also apply the same structural analysis method based on the register cone graph.

Table 4: ML model and training hyperparameters.

Training Phase	**Pre-Training**		**Fine-Tuning**		
ML Model	Graphormer (RTL)	GraphSage (Netlist)	MLP	GCN	XGBoost
# Layers	7	3	2	2	
Hidden Dim	256	256	128	128	
Activation	GELU	ReLU	ReLU	ReLU	100
Batch Size	128		32		estimator,
Optimizer	AdamW		Adam		20
LR	0.001		0.001		max depth
Dataset Size	33162		3278		
# Epochs	75		1000		
Training Time	20h		0.05h		

timing slack on each RTL register facilitates proactive early-stage timing optimization, as demonstrated in [9]. Please note that this task is highly challenging, since the RTL stage contains no physical information necessary for timing evaluation.

(2) RTL-stage overall quality evaluation *at the circuit level*, including worst negative slack (WNS), total negative slack (TNS), and area. This task is benchmarked against SNS v2 [36], MasterRTL [10], and RTL-Timer. Notably, RTL-Timer and MasterRTL utilize task-specific supervised learning, whereas SNS v2 designs a two-phase representation learning model for distinct design quality tasks.

The evaluation metrics of the above two tasks are consistent with those used by the baseline methods, including correlation coefficient (R) and mean absolute percentage error (MAPE).

Functional Reasoning. We evaluate CircuitEncoder at cone-level for netlist-stage functional task: State register identification for netlists *at the cone level*. This task focuses on correctly identifying the control logic and data path registers, which is a critical challenge for reverse engineering [5]. We compare CircuitEncoder with ReIGNN [5], following the label collection process and evaluation metrics, including specificity (i.e., true positive rate) and balanced accuracy (i.e., the average of specificity and true negative rate).

4.3 Evaluation on Downstream Tasks

This subsection highlights the effectiveness of CircuitEncoder on all the above tasks[3], including register slack prediction and state register identification at the cone level, as well as overall WNS/TNS

[3]For the fine-tuning models, we employ MLP or XGBoost for cone-level tasks, and additional GNN layers for circuit level.

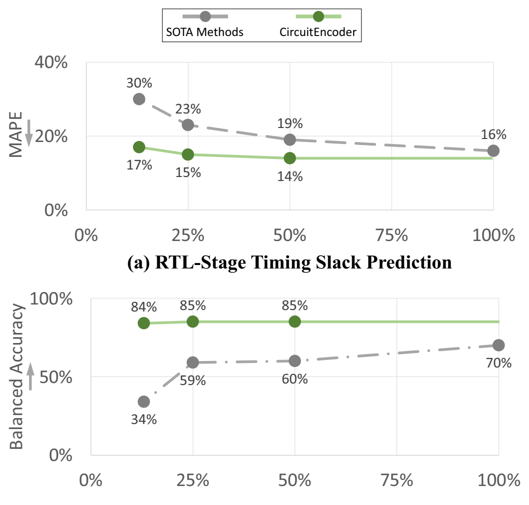


Figure 5: Accuracy comparison when varying the number of datasets for baseline methods training and CircuitEncoder fine-tuning. As the dataset decreases, accuracy drops for supervised methods while CircuitEncoder remains stable.

and area prediction at the circuit level. These cover various highly distinct design quality and functional reasoning tasks for RTL and netlist design stages.

4.3.1 Cone-Level Tasks. We first evaluate CircuitEncoder at the cone level on register timing slack prediction and stage register identification. To highlight its effectiveness, we vary the dataset proportions used for both supervised training of SOTA baseline methods and the fine-tuning of CircuitEncoder. As results shown in Figure 5, reducing the training data significantly decreases the accuracy of baseline methods, whereas CircuitEncoder's performance remains stable due to the self-supervised pre-training.

Table 3 further details the evaluation results of the two cone-level tasks for each test design. After fine-tuning with half of the training designs, CircuitEncoder's few-shot inference significantly outperforms two SOTA methods [5, 9] trained with the full dataset (i.e., MAPE = 14% < 16% and accuracy = 85% > 70%). CircuitEncoder shows robust performance on cone-level tasks. Its effectiveness is attributed to the pre-training that captures the complete state

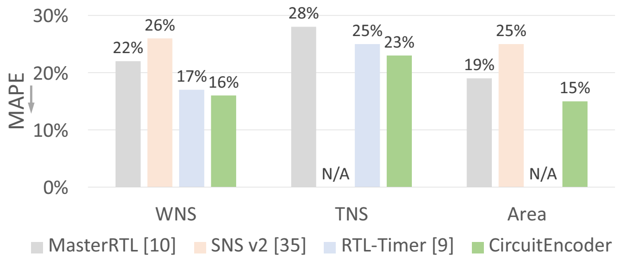


Figure 6: Accuracy comparison for the circuit-level and node-level tasks for RTL and netlist designs.

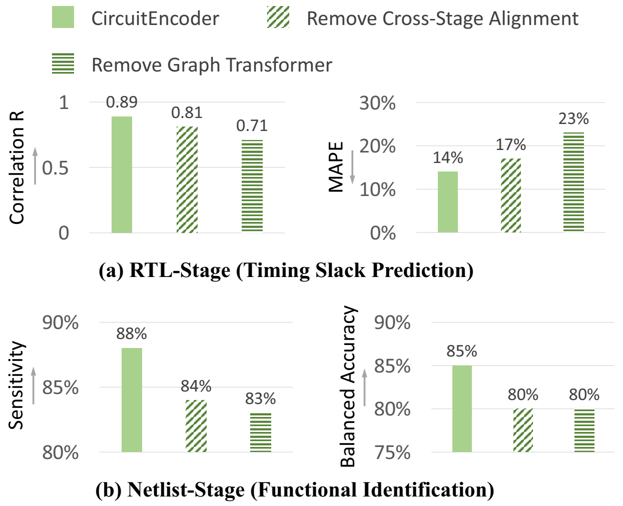


Figure 7: Ablation study. Evaluating the contribution of each strategy used in CircuitEncoder.

transition of each register cone and utilizes cross-stage interactions between RTL and netlist.

4.3.2 Circuit-level Tasks. CircuitEncoder aggregates individual register cones based on their dependency relationships for precise circuit-level modeling. Evaluation results for RTL-stage WNS, TNS, and area predictions are shown in Figure 6. After fine-tuning with 50% of training designs, CircuitEncoder significantly improves accuracy across all three targets. Specifically, it achieves better MAPE in WNS (16% < 17%), TNS (23% < 25%), and area (15% < 19%) predictions than the dedicated supervised methods [9, 10, 36].

4.4 Ablation Study

To evaluate the contribution of various strategies within CircuitEncoder's cross-stage-aligned pre-training process, we conduct ablation studies by selectively removing these key strategies and evaluating both cone-level design quality and functional tasks from Section 4.3.1. The results are summarized in Figure 7. Removing cross-stage alignment by pre-training the RTL and netlist models separately, without incorporating the cross-stage aligned contrastive scheme, results in notable drops in accuracy for both tasks. Additionally, replacing the graph transformer for RTL designs with a GNN leads to significant accuracy drops in both RTL and netlist tasks. This shows that the captured high-level RTL-stage information also benefits the low-level netlists-stage encoding, indicating the effectiveness of cross-stage interactions.

5 CONCLUSION AND FUTURE WORK

In this paper, we present CircuitEncoder, the first self-supervised, pre-trained, and cross-stage-aligned general circuit model. It encodes circuit components into embeddings with rich cross-stage information, based on which lightweight ML models are easily developed for various design quality and functional reasoning tasks. Our future work will focus on improving CircuitEncoder, including supporting more tasks by aligning more design stages, pre-training on more unlabeled circuits, and exploring decoding methods.

6 ACKNOWLEDGMENTS

This work is partially supported by Hong Kong Research Grants Council (RGC) ECS Grant 26208723, National Natural Science Foundation of China (62304192, 92364102), and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

REFERENCES

[1] Christoph Albrecht. 2005. IWLS 2005 benchmarks. In *International Workshop for Logic Synthesis (IWLS): http://www. iwls. org*.
[2] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, et al. 2020. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro* 40, 4 (2020).
[3] Robert Brayton and Alan Mishchenko. 2010. ABC: An academic industrial-strength verification tool. In *Proceedings of 22nd International Conference on Computer Aided Verification (CAV)*. Springer, 24–40.
[4] Lei Chen, Yiqi Chen, Zhufei Chu, Wenji Fang, Tsung-Yi Ho, Yu Huang, Sadaf Khan, Min Li, Xingquan Li, Yun Liang, et al. 2024. The Dawn of AI-Native EDA: Promises and Challenges of Large Circuit Models. *arXiv preprint arXiv:2403.07257* (2024).
[5] Subhajit Dutta Chowdhury, Kaixin Yang, and Pierluigi Nuzzo. 2021. 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, 1–9.
[6] Fulvio Corno, Matteo Sonza Reorda, and Giovanni Squillero. 2000. RT-level ITC'99 benchmarks and first ATPG results. *IEEE Design & Test of computers (ITC)* (2000).
[7] Chenhui Deng, Zichao Yue, Cunxi Yu, Gokce Sarar, Ryan Carey, Rajeev Jain, and Zhiru Zhang. 2024. Less is More: Hop-Wise Graph Attention for Scalable and Generalizable Learning on Circuits. In *Proceedings of 2024 ACM/IEEE Design Automation Conference (DAC)*. ACM, 1–6.
[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[9] Wenji Fang, Shang Liu, Hongce Zhang, and Zhiyao Xie. 2024. 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, 1–6.
[10] Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. 2023. 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, 1–9.
[11] Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. 2024. Transferable Pre-Synthesis PPA Estimation for RTL Designs With Data Augmentation Techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).
[12] Arash Fayyazi, Soheil Shababi, Pierluigi Nuzzo, Shahin Nazarian, and Massoud Pedram. 2019. Deep learning-based circuit recognition using sparse mapping and level-dependent decaying sum circuit representations. In *Proceedings of 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 638–641.
[13] Zizheng Guo, Mingjie Liu, Jiaqi Gu, Shuhan Zhang, David Z Pan, and Yibo Lin. 2022. A timing engine inspired graph neural network model for pre-routing slack prediction. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*. 1207–1212.
[14] Mai Lan Ha and Volker Blanz. 2021. Deep ranking with adaptive margin triplet loss. *arXiv preprint arXiv:2107.06187* (2021).

[15] Zhuolun He, Ziyi Wang, Chen Bai, Haoyu Yang, and Bei Yu. 2021. Graph learning-based arithmetic block identification. In *Proceedings of 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–8.

[16] Sadaf Khan, Zhengyuan Shi, Min Li, and Qiang Xu. 2023. DeepSeq: Deep Sequential Circuit Learning. *arXiv preprint arXiv:2302.13608* (2023).

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

[18] Siting Liu, Qi Sun, Peiyu Liao, Yibo Lin, and Bei Yu. 2021. Global placement with deep learning-enabled explicit routability optimization. In *Proceedings of 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1821–1824.

[19] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. 2021. Self-supervised learning: Generative or contrastive. *IEEE transactions on knowledge and data engineering (TKDE)* 35, 1 (2021), 857–876.

[20] Yikang Ouyang, Sicheng Li, Dongsheng Zuo, et al. 2023. ASAP: Accurate Synthesis Analysis and Prediction with Multi-Task Learning. In *MLCAD*.

[21] Alec Radford, Jong Wook Kim, et al. 2021. Learning transferable visual models from natural language supervision. In *Proceedings of International conference on machine learning (ICML)*. PMLR, 8748–8763.

[22] Prianka Sengupta, Aakash Tyagi, Yiran Chen, and Jiang Hu. 2022. How good is your verilog rtl code? a quick answer from machine learning. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–9.

[23] Zhengyuan Shi, Hongyang Pan, Sadaf Khan, Min Li, Yi Liu, Junhua Huang, Hui-Ling Zhen, Mingxuan Yuan, Zhufei Chu, and Qiang Xu. 2023. Deepgate2: Functionality-aware circuit representation learning. In *Proceedings of IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[24] Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong. 2023. Robust GNN-based representation learning for HLS. In *Proceedings of 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[25] Shobha Vasudevan, Wenjie Joe Jiang, David Bieber, Rishabh Singh, C Richard Ho, Charles Sutton, et al. 2021. Learning semantic representations to verify hardware designs. *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)* 34 (2021), 23491–23504.

[26] VexRiscv. 2022. VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation. https://github.com/SpinalHDL/VexRiscv

[27] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315* (2019).

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

[29] Ziyi Wang, Siting Liu, Yuan Pu, Song Chen, Tsung-Yi Ho, and Bei Yu. 2023. Restructure-Tolerant Timing Prediction via Multimodal Fusion. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[30] Clifford Wolf, Johann Glaser, and Johannes Kepler. 2013. Yosys-a free Verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*.

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

[32] Zhiyao Xie, Yu-Hung Huang, Guan-Qi Fang, Haoxing Ren, Shao-Yun Fang, Yiran Chen, and Jiang Hu. 2018. RouteNet: Routability prediction for mixed-size designs using convolutional neural network. In *Proceedings of 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[33] Zhiyao Xie, Haoxing Ren, Brucek Khailany, Ye Sheng, Santosh Santosh, Jiang Hu, and Yiran Chen. 2020. PowerNet: Transferable dynamic IR drop estimation via maximum convolutional neural network. In *Proceedings of 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 13–18.

[34] Zhiyao Xie, Xiaoqing Xu, Matt Walker, Joshua Knebel, Kumaraguru Palaniswamy, Nicolas Hebert, Jiang Hu, Huanrui Yang, Yiran Chen, and Shidhartha Das. 2021. 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)*.

[35] Ceyu Xu, Chris Kjellqvist, and Lisa Wu Wills. 2022. SNS's not a synthesizer: a deep-learning-based synthesis predictor. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*. 847–859.

[36] Ceyu Xu, Pragya Sharma, Tianshu Wang, and Lisa Wu Wills. 2023. Fast, Robust and Transferable Prediction for Hardware Logic Synthesis. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 167–179.

[37] Zhihao Yang, Dong Li, Yingxueff Zhang, Zhanguang Zhang, Guojie Song, Jianye Hao, et al. 2022. Versatile multi-stage graph neural network for circuit representation. *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)* 35 (2022), 20313–20324.

[38] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do transformers really perform badly for graph representation? *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)* 34 (2021), 28877–28888.

[39] Su Zheng, Lancheng Zou, Peng Xu, Siting Liu, Bei Yu, and Martin Wong. 2023. Lay-Net: Grafting Netlist Knowledge on Layout-Based Congestion Prediction. In *Proceedings of 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[40] Yuan Zhou, Haoxing Ren, Yanqing Zhang, Ben Keller, Brucek Khailany, and Zhiru Zhang. 2019. PRIMAL: Power inference using machine learning. In *Proceedings of 56th Annual Design Automation Conference (DAC)*.

[41] Keren Zhu, Hao Chen, Walker J Turner, George F Kokai, Po-Hsuan Wei, David Z Pan, and Haoxing Ren. 2022. Tag: Learning circuit spatial embedding from layouts. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–9.