CircuitSeer: RTL Post-PnR Delay Prediction via Coupling Functional and Structural Representation

Sanjay Gandham
University of Central Florida
Orlando, FL, USA
sanjay.gandham@ucf.edu

Joe Walston
Synopsys, Inc.
Sunnyvale, CA, USA
joe.walston@synopsys.com

Sourav Samanta
Synopsys, Inc.
Sunnyvale, CA, USA
sourav.samanta@synopsys.com

Lingxiang Yin
University of Central Florida
Orlando, FL, USA
lingxiang.yin@ucf.edu

Hao Zheng
University of Central Florida
Orlando, FL, USA
hao.zheng@ucf.edu

Mingjie Lin
University of Central Florida
Orlando, FL, USA
mingjie.lin@ucf.edu

Stelios Diamantidis
Synopsys, Inc.
Sunnyvale, CA, USA
stelios.diamantidis@synopsys.com

Abstract

Register transfer level (RTL) optimization is a critical design phase that ensures timing closure and performance. Although machine learning (ML) has been utilized to quickly predict post-synthesis delay metrics, estimating post-place and route (PnR) delay remains a significant challenge. This is due to the distinct functionality-preserving characteristics of logic synthesis and the structure-dependent aspect of physical design. Furthermore, Logic Synthesis heavily restructures the netlist, resulting in substantial structural disparities that hinder capturing the post-synthesis netlist structure.

To this end, we present a pre-synthesis delay prediction framework that utilizes deep learning to accurately estimate the post-PnR metrics. Specifically, we propose a graph transformation that extracts synthesis-invariant structural information from our initial circuit representations. Consequently, it allows efficient usage of Graph Neural Networks and introduces the circuit's structural features to the model. Our evaluation of CircuitSeer compared to prior work [1, 2] shows an increase in the R^2 coefficient by 0.44 and a 37% reduction in error. This allows CircuitSeer to be used as a proxy for ASIC implementation in post-PnR delay evaluation.

ACM Reference Format:
Sanjay Gandham, Joe Walston, Sourav Samanta, Lingxiang Yin, Hao Zheng, Mingjie Lin, and Stelios Diamantidis. 2024. CircuitSeer: RTL Post-PnR Delay Prediction via Coupling Functional and Structural Representation . In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24), October 27–31, 2024, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3676536.3676668

1 Introduction

Traditional chip design starts with digital designers meticulously describing the circuits' register-transfer level (RTL). After RTL design, electronic design automation (EDA) tools perform ASIC implementation, which includes Logic Synthesis and Physical Design, to optimize and create the circuit's layout. During this process, quality-of-results (QoR) evalutation can be performed post-synthesis using Static Timing Analysis (STA) tools to generate a preliminary timing report. However, such timing information lacks important cell placement and wire route information leading to a margin of error of up to 60% of the final post-PnR timing information [3]. On the other hand, Physical Design performs structure-dependent operations such as partitioning, floorplanning, placement, and routing on the post-synthesis netlist to create its physical layout. Although post-PnR QoR metrics are the most accurate representation of the design, it comes at a cost of significant runtime as it can take days and weeks for large designs [3] inevitably prolonging the feedback loop as shown in Figure 1.

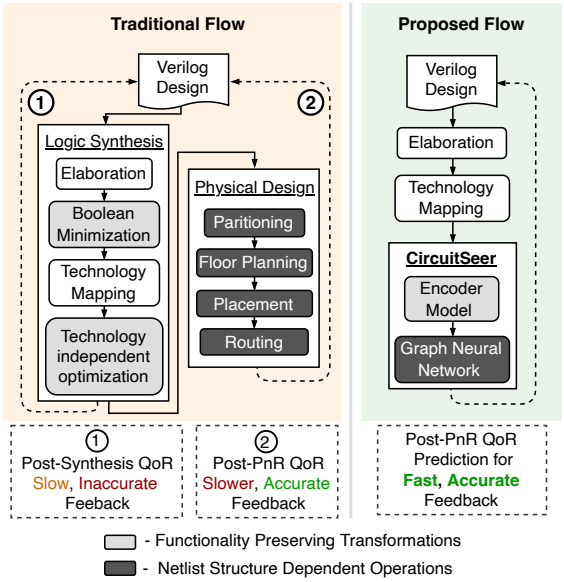

Figure 1: Traditional and Proposed Chip Design Flow with feedback from various stages of ASIC Implementation.

Recently, several studies [1–5] have demonstrated machine learning (ML) model's ability to rapidly estimate a circuit's QoR allowing for fast feedback improving the RTL design process. Moreover, such ML models can be used to drive optimization processes in the EDA Flow [6–11] showing remarkable improvement over traditional heuristic-based optimization. Despite these innovations, prior circuit evaluation models are limited to predicting the QoR metrics

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10
https://doi.org/10.1145/3676536.3676668

Table 1: Comparison with prior QoR evaluation works.

Work	Approach	Receptive Field	Predictive Scope	Output QoR
[1, 2], [4]	Vectorized	Large	Logic Synthesis	Post-synthesis
[3]	Graph	Limited	Physical Design	Post-PnR
Ours	Hybrid	Large	Logic Synthesis & Physical Design	

of a circuit only after either Logic Synthesis or Physical Design (Table 1). For example, [1, 2, 4] estimates the QoR metrics of a circuit after Logic Synthesis from an intermediate representation (IR) of an RTL design. On the other hand, prior work [3] estimates the QoR of a post-synthesis netlist after Physical Design.

However, none of these methodologies estimate the post-PnR QoR from a pre-synthesis circuit representation. The complexity of accurately predicting post-PnR QoR stems from the distinct behavior exhibited in both stages. Logic synthesis aims to create a resource-minimal functionally-equivalent gate representation, netlist, from a verilog design. Meanwhile, physical design relies on the structure of the synthesized netlist to place and route the cells and nets while adhering to physical constraints such as routability and congestion. Therefore, a post-PnR predictive model must capture the functionality-invariant transformations of logic synthesis and netlist structure-dependent operation of physical design. The most straightforward way involves introducing the structural features of the circuit to the post-synthesis QoR prediction models. Previously, Graph Neural Networks (GNN) [12–14] have shown promise in learning a circuit's topological characteristics. Although ideal for capturing the netlist structure-dependent operations of Physical Design [3, 15–17], such methodologies cannot be directly applied to a pre-synthesis delay estimation framework due to the following. The unavailability of a post-synthesis netlist forces such pre-synthesis frameworks to operate on an initial circuit representation. Consequently, the optimizations performed in logic synthesis significantly restructure the netlist, prohibiting efficient structural learning from this initial circuit representation (details in Section 2.2).

To this end, we present CircuitSeer, a novel deep learning (DL)-based prediction framework that can accurately estimate the post-PnR QoR metrics without performing both logic synthesis and physical design directly from a rudimentary technology-mapped netlist. To address the lack of structural similarity between the initial circuit representation and post-synthesis netlist, we propose a series of graph transformations that extract the synthesis-invariant structural information from our initial circuit representation. CircuitSeer uses a message-passing GNN on this transformed graph to capture the netlist topological features, significantly improving its predictive accuracy as shown by extensive ablation studies. Our major contributions are summarized as follows:

- To the best of our knowledge, this is the first pre-synthesis QoR estimation framework that predicts design delay metrics after Logic Synthesis, Placement, and Routing.
- We propose graph transformations that improve the structural similarity between the RTL representation and post-synthesis netlist, enabling GNNs for structural learning.
- We present a novel DL architecture using attention-based encoders and Graph Neural Networks to generate functional and structural circuit representations of a netlist, enabling downstream QoR Evaluation tasks.
- By evaluating CircuitSeer using ablation and comparison studies, we observe a 0.44 improvement in R^2 coefficient and 37% reduction in mean average percentage error compared to prior work [1, 2] allowing it to act as a proxy to ASIC implementation for fast post-PnR delay evaluation.

Table 2: Cell distribution of various designs obtained after tech-mapping and post-synthesis.

Design [18]	Combinational Cells			Sequential Cells		
	N_u	N_{ps}	ΔN^*	N_u	N_{ps}	ΔN^*
picorv32	2853	5049	76.97%	1727	1728	0.06%
dma	2941	4235	43.99%	1919	1924	0.26%
wb_dma	890	1127	26.63%	539	541	0.37%
i2c	248	322	29.83%	138	138	0.00%
wb_conmax	12091	14951	23.65%	858	858	0.00%

N_u = Unoptimized Netlist; N_{ps} = Post-Synthesis Netlist
$*$: Lower values indicate high similarity.

2 Preliminaries

2.1 Circuit Representation for QoR Evaluation

Prior pre-synthesis QoR prediction frameworks typically convert the input verilog into an Intermediate Representation (IR) like an Abstract Syntax Tree (AST) [1, 2] or a Simple Operator Graph (SOG) [4]. This IR is integrated into ML models either by vectorizing parts of it or by directly operating on it. Although showing promising capabilities, such methodologies struggle with transferability between standard cell libraries. Logic synthesis is performed based on the availability of standard cells in a cell library. For example, logic synthesis might use a large 8-input `OR` gate to minimize area or decompose it into smaller 2-input `OR` gates to optimize delay. Such optimizations are inherent to cell libraries. Consequently, an ML model trained with a library-invariant IR as an input could have limited applicability when the cell library is changed.

To remedy this issue, we propose converting the RTL design to a library-specific input by only performing elaboration and technology mapping. By operating on a tech-mapped netlist, the model can differentiate the cells of various libraries allowing for simultaneous learning of multiple technology nodes in the same model. In this paper, we refer to the netlist obtained after elaboration and techmapping as an unoptimized netlist N_u. Note that the unoptimized netlist N_u is not the same as a post-synthesis netlist N_{ps}. While logic synthesis performs significant optimizations to obtain the post-synthesis netlist N_{ps}, no optimizations have been applied to the unoptimized netlist N_u. MasterRTL [4] attributes the improved predictive capabilities of SOG compared to AST to the similarity and consistency in register mapping between SOG and post-synthesis netlist. Similarly, the unoptimized netlist N_u also retains such desirable characteristics (Section 3.3) while allowing for model training on diverse technology libraries.

2.2 Unoptimized Netlist Evaluation

To understand the difference between the unoptimized netlist N_u and post-synthesis netlist N_{ps}, we analyze the netlists for various verilog designs obtained from IWLS benchmarks [18], OpenCores [19], and open-source RISC-V processors. Table 2 shows a

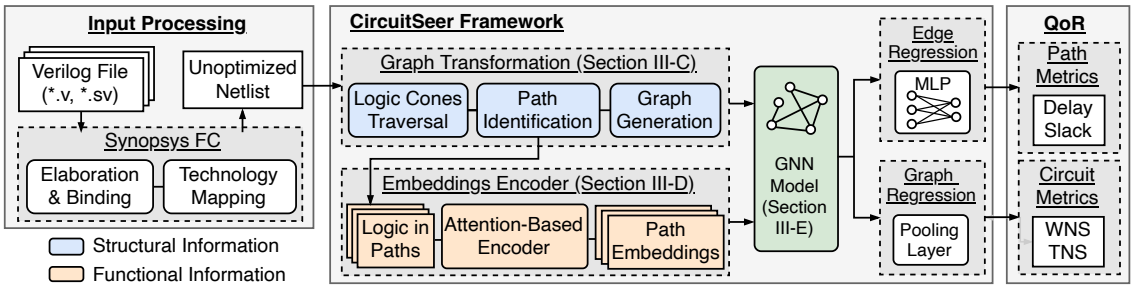

Figure 2: Overview of CircuitSeer prediction flow.

breakdown of combinational and sequential cells of both netlists obtained from Synopsys Fusion Compiler [20]. Here, ΔN represents the difference in cell count between netlists N_{ps} and N_u. The difference, ΔN, of combinational cells is significant with an average increase of 40.21% in combinational cells for N_{ps} compared to N_u. This is due to the designs being constrained to operate at maximal frequency, minimizing logic sharing between logic cones. Furthermore, logic synthesis breaks down complex cells into multiple simpler cells for lower timing delays. As a result, the number of combinational cells is increased. This difference in cell count indicates that the netlist N_{ps} is not structurally similar to N_u.

Although the combinational cells differ between the two netlists, Table 2 also shows the number of sequential cells remains nearly unchanged with minimal ΔN values. We observe that, while the combinational logic in the fan-in logic cone is altered, the sequential startpoints and endpoints are roughly preserved. Based on this observation, we propose a transformation methodology (Section 3.3) that converts the netlist N_{ps} to a graph of sequential cells, increasing the structural similarity between the netlists N_u and N_{ps}.

2.3 Graph Neural Networks for EDA

Prior works [3, 17, 21–26] have demonstrated that Graph Neural Networks (GNNs) [13, 27] can effectively address challenging EDA problems due to their ability to operate on graph data [26, 28, 29]. GNN-based methodologies have been applied in design analysis, such as power estimation [21], and design optimizations including gate sizing [23]. However, such GNN models cannot be directly used for post-synthesis structural learning on pre-synthesis circuit representation for the following reasons.

1) The lack of structural similarity between a pre-synthesis representation of a circuit and the post-synthesis netlist. Prior work [30–32] has shown GNNs' tendency to capture the structural characteristics of the netlist. However, logic synthesis involves optimizations that modify the netlist's logic and structure(e.g., Boolean restructuring, gate decomposition, etc.). Such optimizations retain the circuit's functionality but replace gates or rewire the connections between existing gates to improve driving strength and signal integrity. Therefore, using GNNs directly on a pre-synthesis circuit representation, which differs structurally from the post-synthesis netlist, isn't ideal for structural representation learning.

2) GNNs typically suffer from a limited respective field, which determines the distance of information propagation in the graph. Prior work [25] has shown that an increased receptive field, achieved by using a GNN model with more layers, improves performance in EDA tasks. However, to properly capture the circuit's structural information, such as logic cones' fan-in characteristics that typically span over tens and hundreds of combinational logic cells, traditional GNNs must have a receptive field beyond their capabilities [12]. In this paper, we aim to explore the efficient application of ML models, including GNNs, on a pre-synthesis framework for post-PnR QoR prediction while bypassing their inherent limitations.

3 CircuitSeer Methodology

3.1 Problem Statement

In this work, we develop a path-based QoR prediction framework that can estimate the post-PnR delay $T_{P_{i\to j}}$ of logic paths $P_{i\to j}$ in an RTL design where $P_{i\to j} \in R_N$. R_N is a set of valid logic paths in a netlist N and i, j are startpoint, endpoint pairs from the set of I/O and sequential elements of the RTL design. While circuit-wide global QoR is typically reported, we aim to report a localized QoR as design-wide QoR such as the worst negative slack (WNS), and total negative slack (TNS) can be inferred from an aggregation of local QoR. With T_t as the design's target delay, WNS and TNS can be represented as follows

$$\begin{aligned} WNS &= \min_{P_{i\to j}\in R_N} \{T_t - T_{P_{i\to j}}\} \\ TNS &= \sum_{P_{i\to j}\in R_N} (T_t - T_{P_{i\to j}}) \end{aligned} \tag{1}$$

Besides, local information is more useful as it can be used as precise feedback for RTL designers or optimization frameworks to further improve the QoR. We formulate the pre-synthesis QoR evaluation problem as follows. Given an Unoptimized Netlist of an RTL design as N_u whose Post-Synthesis Netlist is N_{ps}, we develop a post-PnR QoR prediction framework F that evaluates metrics of a path $P_{i\to j}$ without Logic Synthesis and Physical Design. It can be summarized as below.

$$F(N_u) \to \{T_{P_{i\to j}}\}, \forall P_{i\to j} \in R_{N_{ps}}$$

3.2 CircuitSeer Overview

The key challenge in introducing netlist structural features to the model lies in the structural dissimilarity between the initial circuit representation and post-synthesis netlist. To bypass this, we propose a framework that extracts synthesis-invariant structural information and facilitates structural learning on the initial circuit representation. Figure 2 shows the overview of CircuitSeer's methodology for the post-PnR localized delay prediction from an input verilog design. First, we perform elaboration and technology mapping of an input verilog design to obtain an unoptimized netlist,

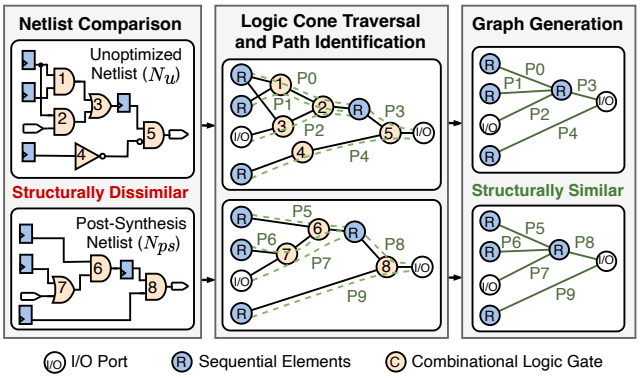

Figure 3: Graph transformation on unoptimized and optimized netlists to improve structural similarity.

N_u, that is converted to a graph representation. Then we take a three-step approach to facilitate model inference.

(1) First, we identify all the logic cones, the region of combinational logic in the design with one sequential element as the endpoint and all other connecting sequential elements as starting points. Later, we use an encoder model to generate a concise representation of the logic between each of the starting point and endpoint pairs to capture the optimizations performed by Logic Synthesis.
(2) Next, we perform graph transformations on the netlist N_u to increase the structural similarity between the pre-synthesis design representation and the post-synthesis netlist N_{ps}. The transformed graph represents sequential elements as vertices and combinational logic between each timing path as an edge.
(3) Finally, we annotate the encoder-generated embeddings on the edges of the transformed graph and use a GNN model to infer the complex structural interactions of logic as exhibited in Physical Design.

The proposed methodology can facilitate multiple downstream tasks, including but not limited to edge regression for path-level delay and graph regression for circuit-level WNS and TNS. In this work, we primarily focus on path-level delay estimation.

3.3 Graph Transformation

The inability to use GNNs for structural learning on an unoptimized netlist N_u lies in the lack of structural similarity between the netlist N_u and the post-synthesis netlist N_{ps}. Figure 3 shows two functionally identical but structurally different netlists. The unoptimized netlist, N_u did not undergo boolean minimization. As a result, such a preliminary representation of the circuit contains redundant logic gates and inefficiently represents the circuit function. The post-synthesis netlist, on the other hand, represents the same circuit with fewer logic gates optimizing for timing, area, and power costs. Such a restructured netlist prohibits structural learning, a key factor in determining delay-modifying operations (buffer insertions, placement, routing, etc.) in physical design [3]. As such, we propose a transformation, $T(N)$, on a netlist N that converts it to a graph representation, G, such that the transformed unoptimized netlist $T(N_u) = G_u$, and the transform post-synthesis netlist $T(N_{ps}) = G_{ps}$ have a high degree of structural similarity.

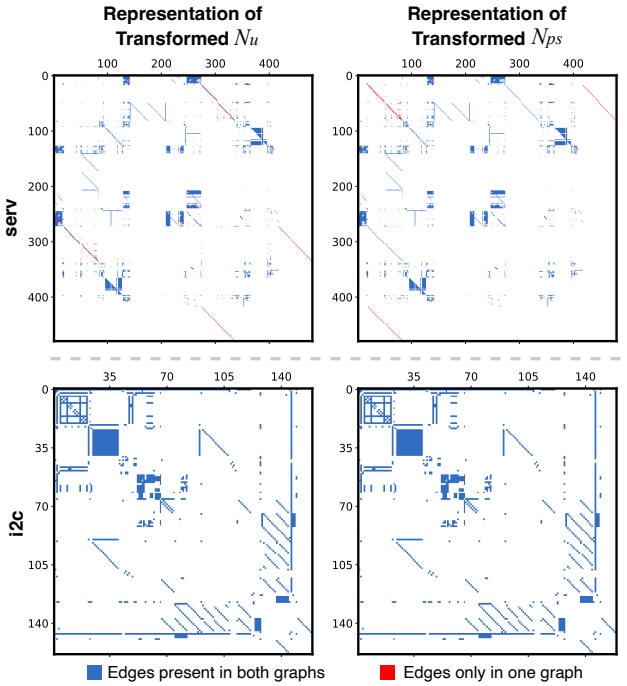

Figure 4: Adjacency matrix representation of transformed unoptimized and post-synthesis netlist of real-world designs: serv and i2c.

The proposed transformation retains the desired structural characteristics of the netlist N_{ps}. Our evaluation of CircuitSeer later shows that introducing such structural features improves the models' predictive capabilities. We use Figure 3 as an example to explain the proposed transformation on netlists N_u and N_{ps} and show an improved structural similarity between their transformed graph representation G_u and G_{ps}. Our proposed graph transformation involves the following steps.

3.3.1 Logic Cone Traversal and Path Identification. First, we isolate each circuit's sequential (registers and IO ports) element and identify its logic cone. To perform this, we start from the sequential endpoint and utilize depth-first search to traverse the netlist until a sequential startpoint is reached. Each sequential element logic cone traversal is terminated after the depth-first search has visited all the edges of the logic cones' combinational elements. For netlist N_u, after identifying the first path $P0$, our proposed search algorithm will backtrack until it finds a combinational cell with an unvisited edge (i.e., logic gate 1). By continuing this algorithm, we identify 5 paths for the netlist N_u as shown in Figure 3.

3.3.2 Graph Generation. During the logic cone traversal, we record the logic between each sequential startpoint and endpoint pair. While multiple paths may exist between the same startpoint and endpoint pair, we retain the path with the most logic cells as they typically dictate the path delay. This recorded path information is later sent to the encoder (described in section 3.4) for functional representation learning. Based on the identified paths, we generate a graph where each vertex represents a sequential element, and each edge corresponds to a logic path between the sequential elements, shown in the rightmost part of Figure 3. In this example, the

Table 3: Vertices and edges comparison between the transformed unoptimized and transformed Post-Synthesis netlist.

Design	Transformed UO Netlist G_u (Vertices, Edges)	Transformed PS Netlist G_{ps} (Vertices, Edges)	Vertex Match (%)	Edge Match (%)
i2c	160, 1856	160, 1856	100	100
serv	481, 6021	481, 5988	100	94.02
picorv32	1846, 71681	1846, 71657	100	97.46
ac97_ctrl	2304, 17959	2304, 17959	100	99.99

generated graph structures for netlists N_u and N_{ps} are exactly the same. We further evaluate the efficacy of such transformations on real-world designs taken from RISC-V cores and the IWLS benchmark. To visualize the similarity between the graphs, we present their adjacency matrices. The adjacency matrix is an $V \times V$ matrix, where each row and column indicates the vertex (i.e., the sequential element) index and the colored pixel indicates the presence of an edge (i.e., the path) between the corresponding row and column vertices. Figure 4 shows the adjacency matrices for two real-world designs: serv and i2c. While the matrices for i2c show that the transformed netlist G_u and G_{ps} are equivalent, there is a slight difference (less than 6%) between them for serv, indicated by the red dots. This discrepancy is due to register retiming and duplication, an optimization performed by the logic synthesis tool that changes the connectivity of sequential elements to meet timing closure. Although such optimizations can potentially change these edges of the transformed graph, most of the edges are preserved. Table 3 shows the detailed number of edges and vertices in the transformed graphs G_u and G_{ps}. Vertex and Edge match indicates the percentage of vertices and edges in the transformed unoptimized graph G_u that is retained in the transformed post-synthesis graph G_{ps}. We observe that all vertices are preserved between the two netlists, while over 94% of edges are retained, and such a transformed graph retains the synthesis-invariant structural information. We later show that introducing such structural information improves the graph neural network model's predictive capabilities in subsequent sections.

3.4 Encoder for Path Embeddings

The transformed graph G_u consolidates the combinational cells of a logic path into an edge between the startpoint and endpoint sequential elements (vertices). To properly reflect the timing characteristics of a sequential endpoint, the information within the logic cone must be represented concisely. However, the depth of timing paths in a logic cone can range from a few cells to hundreds of cells making such representation difficult. Borrowing from Natural Language Processing models where a variable-length text is captured by a fixed-dimension representation, we use an encoder mechanism [33] to convert a variable-length logic path into a fixed-dimension embedding. The GNN model later uses this embedding to perform structural representation learning. As shown in Figure 5, we first generate a vocabulary of library-specific logic gates where we assign an integer to each type of cell. Then, we receive the logic paths which are a sequence of library-specific cell names $\{C_1, C_2, ...C_n\}$ in the paths $P_{i \to j} \in N_u$ obtained from the logic cone traversal of netlist N_u. Based on the generated vocabulary, we vectorize the cell sequence of each path and feed it to an attention-based encoder. Prior work [1, 34] employs a similar approach. For example, the

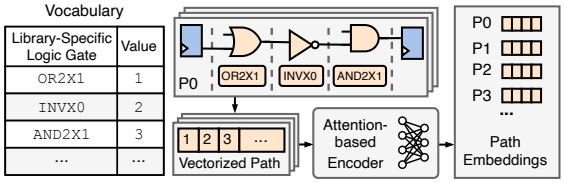

Figure 5: Encoder embedding flow.

three cells in the Path $P0$ are represented by `'OR2X1'`, `'INVX0'` and `'AND2X1'` are vectorized to values 1, 2, and 3. The vectorized paths are used by the attention-based encoder to generate concise path embedding representing the logic paths. These embeddings are used to annotate the graph edges for downstream GNN processing.

3.5 GNN Model

Prior to the graph learning process, we utilize the encoder-generated embedding for each logic path as edge features e_{uv} between source and destination vertex u, and v as shown in Figure 6. Similarly, we initialize the vertex features h_u to zero vectors, setting a starting point for structural feature learning. Although GNNs have been used for mining circuit structural information, we customize a GNN variant to properly reflect the netlist structural interactions. We aim to embed the netlist structural information of the local neighborhood of the sequential elements in the graph vertices. During the process, for each edge, the source vertex u generates a message to be sent to the destination vertex v. Based on the fact that a wider logic cone with multiple sequential startpoints is indicative of congestion, we use a sum aggregator to consolidate the messages. Such an aggregator introduces features that correspond to the topography around a sequential element. These aggregated messages are further transformed using a linear layer with learned weight parameters as shown in Equation 2.

$$m_v^k = \sum_{u \in N(v)} W_u^k [h_u^{k-1} || e_{uv}] \tag{2}$$

Here m_v^k is the aggregated message for destination vertex v at layer k, $N(v)$ represents the neighbors of v, h_u^{k-1} are the features of source vertex u from the previous layer, W_u^k is the weight parameter, and $||$ denotes the column-wise concatenation. After receiving the aggregated message m_v^k from its neighbors, the destination vertex v then updates its features for the next layer, as described by Equation 3.

$$h_v^k = \sigma \left(W^k [m_v^k || h_u^{k-1}] \right) \tag{3}$$

During this process, the vertices learn the topological characteristics of their local neighborhood. Furthermore, unlike a cell-level graph representation, operating on a path-level graph representation enhances the limited receptive field of GNNs allowing for improved structural representation learning.

3.6 Edge Regression

During the n-layer graph learning, the vertices learn the structural characteristics of the logic cones from its n-hop neighborhood. Then, we perform edge regression on each edge (logic path) to estimate its post-PnR delay. We couple the functional representation with the structural information by concatenating the edge features

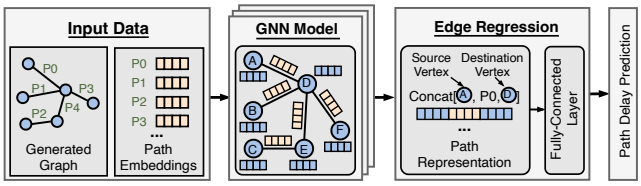

Figure 6: GNN inference and edge regression flow.

with the learned source and destination vertex features. Figure 6 demonstrates the concatenation of features from vertex A, path $P0$, and vertex D to obtain the final representation for Path $P0$. Afterward, we use a Multi-Layer Perceptron (MLP) to perform the edge regression described in Equation 4.

$$T_{P_{i\to j}} = MLP\left([h_u^n || e_{uv} || h_v^n]\right) \tag{4}$$

3.7 Model Training Methodology

To train the CircuitSeer model, we curate a set of unoptimized netlists N_u, and the ground truth QoR for paths $P_{i\to j} \in R_{N_{ps}}$. CircuitSeer first identifies paths $P_{i\to j} \in R_{N_u}$, then for each ($P_{i\to j} \in R_{N_u}, N_u$) tuple, we infer the path QoR $T_{P_{i\to j}}$. We compare the predicted values with the ground truth, calculate the mean-squared error, and use an Adam optimizer [35] with a learning rate of 1e−6 to backpropagate the error through the edge regression, GNN, and the encoder model. Prior work [4] has shown the scarcity of circuit information for model training. However, such methodologies use a per-design training approach requiring numerous circuit designs. On the other hand, we employ a path-based training approach where we isolate timing paths and their local neighborhood structure from the circuit. This uncovers numerous training samples obviating the need for dataset augmentation with synthetic data. Table 4 shows that only 18 circuits lead to 403,039 unique samples.

4 Evaluation

4.1 Experimental Setup and Datasets

We implement CircuitSeer using PyTorch-Geometric [36] library for the 2-layer GNN model, and the mobileBERT encoder from the HuggingFace library and a 2-layer fully connected neural network as the Multi-Layer Perception (MLP). We use an encoder model with 2 attention heads, embedding dimension of 32, and intermediate size of 64. For the GNN model, we use a feature dimension of 32. We train and evaluate the model on a server with two NVIDIA H100 GPUs with a batch size of 64 paths for 180 epochs.

Dataset Generation. We utilize 18 circuits with varying design characteristics from popular benchmark suites: ITC'99 [37], IWLS'05 [18], OpenCores [19], and RISC-V cores. Table 4 shows the designs, their combinational and sequential cell count, and path delay statistics used for training and testing CircuitSeer. We split the designs into 14 training and 4 testing datasets with 364,114 training and 38,925 testing paths. Note that the test designs are completely unseen designs, mimicking CircuitSeer's real-world usage scenario. We employ Synopsys Fusion Compiler [20] with SAED32_HVT 32nm standard cell library to perform elaboration and technology mapping to generate the unoptimized netlist. Although Synopsys Fusion Compiler was used, any Logic Synthesis tool such as Yosys [38] can also generate the unoptimized netlist. We further collect the ground truth path delays after running the entire ASIC implementation using Synopsys Fusion Compiler *compile_fusion* flow constrained to its maximum operational frequency.

Table 4: Breakdown of various training and testing circuit designs on cell count, path delay, and number of paths.

Training Designs	Cell Count		Path Delay (*ns*)	# Paths
	Seq.	Comb.	{min, avg, max}	
i2c	138	248	{0.00, 3.59, 6.43}	1,104
s13207	221	387	{0.00, 3.23, 5.77}	1,558
s5378	163	464	{9.45E-2, 4.98, 8.28}	2,419
ac97_ctrl	2,283	1,766	{1.90E-5, 3.61, 6.55}	9,083
s35932	1,472	2,589	{1.25E-1, 3.11, 7.56}	9,387
des_perf	1,984	10,967	{0.00, 10.31, 13.65}	11,280
b18	800	3,474	{1.18E-1, 8.71, 18.9}	16,090
mem_ctrl	1,096	1,878	{0.00, 9.09, 13.53}	24,117
pci_bridge32	3,335	3,013	{0.00, 7.14, 13.65}	25,313
Faraday_dma	1,919	2,941	{0.00, 6.6, 11.38}	30,791
s38417	1,396	2,486	{0.00, 9.55, 14.84}	31,533
picorv32	1,727	2,853	{0.00, 7.66, 12.58}	36,077
wb_conmax	858	12,091	{1.02E-1, 8.03, 15.35}	55,035
ethernet	11,692	9,512	{0.00, 8.36, 15.45}	71,402
Testing Designs	**Seq.**	**Comb.**	**Path Delay (*ns*)**	**# Paths**
serv	195	563	{1.90E-5, 6.18, 9.38}	3,034
s38584	862	1,856	{8.33E-2, 4.09, 9.39}	6,628
wb_dma	539	890	{0.00, 5.06, 7.55}	8,375
usb_funct	1,805	3,369	{1.68E-3, 4.43, 7.1}	20,888

Baseline Framework. Since this is the first work to estimate the post-PnR delay metrics from a pre-synthesis circuit representation, we choose to use the SOTA post-synthesis delay prediction frameworks [1, 2] as our baselines. Although they operate on an AST representation, for a fair comparison with CircuitSeer, we train their constituent Transformer and XGBoost model on all the paths obtained from the unoptimized netlist with post-PnR delay metrics as the ground truth. While baselines [1, 2] evaluate circuit-wide delay metrics such as WNS and TNS, we repurposed the models to infer path delay metrics. Due to the bit-blasted representation of the unoptimized netlist N_u compared to the AST representation, the inputs to the baselines are the logic cells of each path, either as a sequence or in its tabular form. We modify the vocabulary and embedding dimensions to match CircuitSeer. Due to the extensive changes performed to each of the models used in prior work [1, 2], we refer to the baselines with their underlying Transformer [33] and XGBoost model [39].

Evaluation Metrics. We use R Correlation Coefficient, R^2 Determination Coefficient, and Mean Absolute Percentage Error (MAPE), to evaluate the estimation accuracy between the ground truth and predicted values of the testing designs.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|x_i - y_i|}{x_i} \times 100\%$$

$$R = \frac{\Sigma_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\Sigma_{i=1}^{n}(x_i - \bar{x})^2 (y_i - \bar{y})^2}}$$

$$R^2 = 1 - \frac{\Sigma_{i=1}^{n}(y_i - x_i)^2}{\Sigma_{i=1}^{n}(y_i - \bar{y})^2}$$

The y and x are the predictions and ground truth values. Whereas $\bar{y}$ and $\bar{x}$ are the mean values of y and x respectively. A lower value

Table 5: Accuracy Comparison using R^2 coefficient, mean absolute error percentage (MAPE) of CircuitSeer and baselines on unseen test circuits. Lower MAPE and higher R^2 indicate better prediction accuracy.

Model	serv			wb_dma			usb_funct			s38584		
	R^2	MAPE 50	MAPE 25	R^2	MAPE 50	MAPE 25	R^2	MAPE 50	MAPE 25	R^2	MAPE 50	MAPE 25
Transformer [1]	0.38	0.20	0.18	-0.40	0.35	0.34	0.23	0.28	0.23	0.54	0.26	0.20
XGBoost [2]	0.43	0.19	0.19	-0.22	0.30	0.29	-0.07	0.31	0.26	0.37	0.30	0.29
CircuitSeer (CS)	**0.66**	**0.14**	**0.11**	**0.38**	**0.22**	**0.18**	**0.67**	**0.17**	**0.16**	**0.79**	**0.17**	**0.15**

MAPE K: indicates the Mean Average Percentage Error for the longest K% paths of the design.

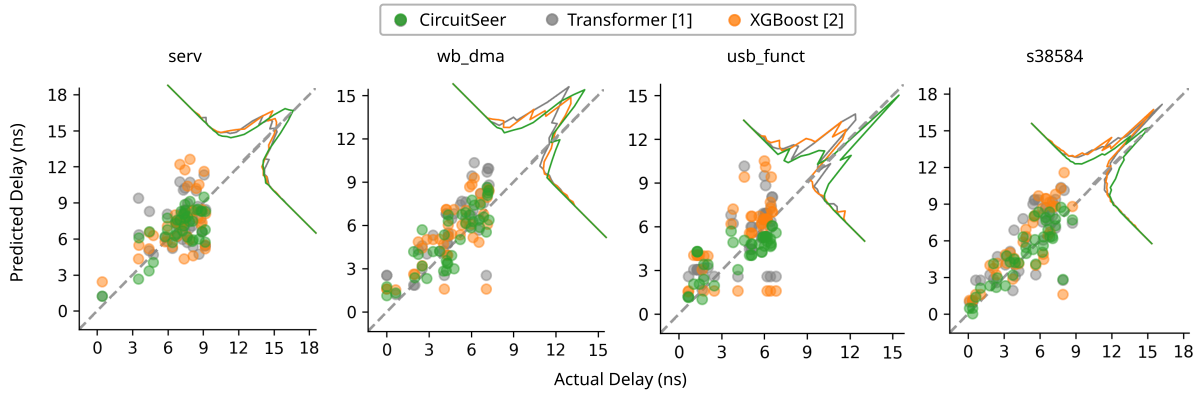

Figure 7: Scatter plot comparing the predicted delay and actual delay for all designs across test circuits. The histogram shows the distribution of prediction errors relative to the ground truth (the dashed line). CircuitSeer significantly outperforms prior SOTA models [1, 2].

of MAPE and a higher value of R and R^2 indicate higher model accuracy and well-correlated predictions.

4.2 Comparative Analysis of Circuit Delay Prediction Models

Table 5 presents a comparative analysis of CircuitSeer against baseline models using R^2 and Mean Average Percentage Error (MAPE) metrics for the 50% longest paths (MAPE 50), and for the 25% longest paths (MAPE 25). CircuitSeer consistently achieves higher R^2 values and lower MAPE scores across all circuits, indicating superior predictive accuracy. Specifically, CircuitSeer's lower average MAPE 50 of 17.5% compared to 27.3% and 27.5% of Transformer and XGBoost highlights its enhanced accuracy in delay prediction. Moreover, CircuitSeer shows an improved average R^2 correlation of 0.625, significantly exceeding the 0.1875 and 0.1275 achieved by Transformer and XGBoost, indicating that CircuitSeer's predictions are highly correlated with actual post-PnR circuit path delays. Interestingly, the R^2 Determination Coefficient of Transformer model for *wb_dma* and XGBoost for both the *wb_dma* and *usb_funct* circuits are negative. This indicates that these models perform worse than a constant function that predicts the mean of the data. Such a poor predictive performance indicates that the circuit's structural complexity significantly affects its post-PnR delay metrics for *wb_dma* and *usb_funct*. Nonetheless, CircuitSeer consistently shows a stronger correlation, demonstrating its effectiveness and robustness in handling varying circuit complexities.

4.3 Visualization of Path Delay Predictions

Figure 7 shows the scatter plot of CircuitSeer and baseline for each test circuit. Each colored dot corresponds to a logic path with its predicted delay plotted against the actual delay in nanoseconds (*ns*) on the Y and X axes, respectively. Dots closer to the diagonal line represent a more accurate prediction. For consistency, we perform delay prediction for the same paths across all the baselines. Towards the top-right corner of each scatter plot, we plot the histogram showing the distribution of prediction errors relative to the diagonal. The Transformer and XGBoost model's distributions deviate significantly from the diagonal, characterized by broader and flatter peaks, indicative of larger variance in their predictions. Conversely, CircuitSeer's histogram maintains a narrower profile with peaks that closely align with the diagonal, reflecting its accuracy and reduced variance. For a more detailed comparison, Figure 8 shows the heatmap for each prediction across the baselines. We generate the heatmap by binning the actual and predicted data into axis-aligned bins with a width and height of 0.5*ns*. The darker dots in the heatmap represent a higher concentration of points in the bin. We observe a higher density near the diagonal line for CircuitSeer, compared to baselines, particularly for longer path delays, which are critical for optimizing circuit performance. Accurate prediction of these delays, such as WNS, is essential, as they often dictate the overall circuit performance. In contrast, the Transformer and XGBoost show a larger vertical banding away from the diagonal, particularly towards the longer paths, indicating lower prediction accuracy. On the other hand, CircuitSeer demonstrates a much tighter clustering with points closer to the diagonal. This observation is corroborated by CircuitSeer's lower MAPE 25 as shown in Table 5, making CircuitSeer ideal for critical path delay estimation.

4.4 Runtime Analysis

We evaluate the runtime differences between a complete ASIC implementation using Synopsys Fusion Compiler and our CircuitSeer flow for path-level QoR estimation under maximal operational

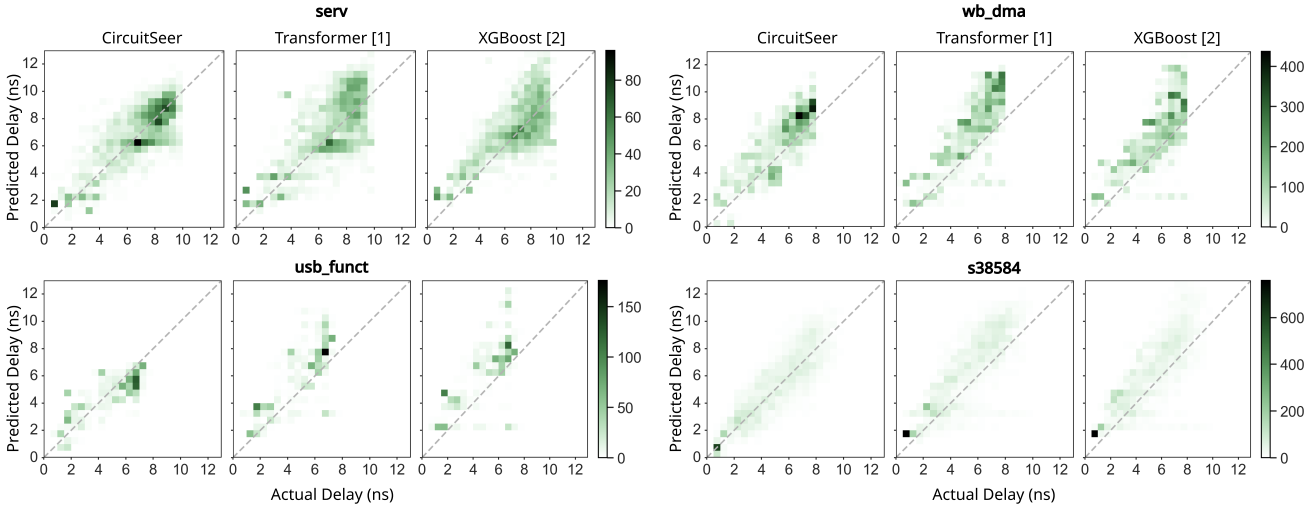

Figure 8: Heatmaps for predicted vs actual delay across test circuits using different models. Darker colored dots indicate higher concentration and dots closer to the dashed line represent lower errors.

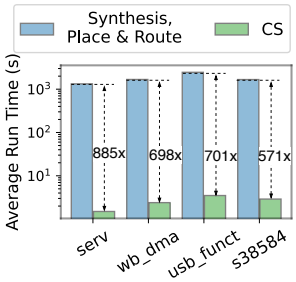

Figure 9: Run time comparison with the traditional flow.

Table 6: Runtime breakdown of CircuitSeer.

Execution Phase	Runtime (%)
Elaboration & Tech. Mapping	93.4
Logic Cone Traversal	5.2
Graph Generation	0.3
Model Inference	1.1

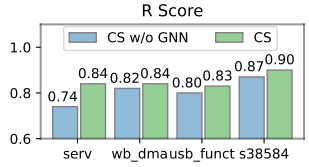

Figure 10: Comparison of R score for CS without GNN and CS for the ablation study.

Table 7: Comparison of MAPE 25 for CS without GNN and CS for the ablation study.

Design	CS w/o GNN	CS
serv	18.5%	**11.3%**
wb_dma	34.5%	**18.2%**
usb_funct	23.4%	**15.9%**
s38584	19.7%	**15.2%**

frequency constraints. ASIC implementation typically involves multiple iterative runs to fine-tune delay constraints to achieve the circuit's maximal operational frequency. In our testing, we performed 10 runs with progressively adjusted delay constraints reporting the cumulative time taken by the Synopsys Fusion Compiler. Unlike traditional methods that require synthesizing, placing, and routing the entire design for path-level delay evaluation, CircuitSeer targets a localized subgraph that falls within the receptive field of the target path. This process involves iterating over the logic cones connected to the start and end points of the target path. Consequently, we compare the total time taken for complete ASIC implementation to the localized QoR estimation performed by CircuitSeer. As shown in Figure 9, CircuitSeer demonstrates an average speedup of 713× compared to traditional ASIC implementation. Table 6 shows a breakdown of time spent by CircuitSeer in all its constituent phases. Elaboration & Tech-mapping takes 93.4% of the overall runtime time. It's important to note that the time reported for elaboration and tech-mapping pertains to the entire circuit. Since CircuitSeer's operations are limited to the local neighborhood of a timing path, restricting elaboration and tech-mapping to just this neighborhood could significantly reduce CircuitSeer's execution time.

4.5 Ablation Study

To assess the impact of introducing topological features on the model's predictive accuracy, we perform an ablation study by training an identical version of CircuitSeer without its GNN component. In this variant, we utilized only the encoder to generate path embeddings, which were then processed by a Multi-Layer Perceptron (MLP) to perform path-wise regression directly. Figure 10 illustrates the R score across all the test designs. The results indicate a higher average R score of 0.852 for the complete CircuitSeer model, compared to 0.807 for the version without GNN. Additionally, Table 7 shows that CircuitSeer (CS) achieves a lower average MAPE 25 value of 15.2%, compared to 24.0% observed for the model without GNN. This improvement in predictive performance shows that introducing the topological feature using GNNs improves the model's predictive capabilities. Furthermore, the successful introduction of structural features is due to the proposed graph transformations that improve the structural similarity between the unoptimized netlist and the post-synthesis netlist.

5 Conclusion

We present CircuitSeer, a pre-synthesis delay prediction framework to accurately estimate the post-PnR metrics. Specifically, we propose a graph transformation that extracts synthesis-invariant structural information from our initial circuit representations. Consequently, it allows efficient usage of Graph Neural Networks and introduces the circuit's structural features to the model. Our evaluation of CircuitSeer compared to prior work [1, 2] shows an increase in the R^2 coefficient by 0.44 and a 37% reduction in error. This demonstrates that CircuitSeer can effectively serve as a proxy for ASIC implementation in post-PnR delay estimation.

References

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

[2] Prianka Sengupta, Aakash Tyagi, Yiran Chen, and Jiang Hu. 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)*, pages 1–9. IEEE, 2022.

[3] Ahmed Agiza, Rajarshi Roy, Teodor Dumitru Ene, Saad Godil, Sherief Reda, and Bryan Catanzaro. Graphsym: Graph physical synthesis model. In *Proceedings of the 42nd IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.

[4] Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. Masterrtl: A pre-synthesis ppa estimation framework for any rtl design. In *Proceedings of the 42nd IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.

[5] Prianka Sengupta, Aakash Tyagi, Yiran Chen, and Jiang Hu. Early identification of timing critical rtl components using ml based path delay prediction. In *Proceedings of the 5th Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6. IEEE, 2023.

[6] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. Chip placement with deep reinforcement learning. In *arXiv preprint arXiv:2004.10746*, 2020.

[7] Rajarshi Roy, Jonathan Raiman, Neel Kant, Ilyas Elkin, Robert Kirby, Michael Siu, Stuart Oberman, Saad Godil, and Bryan Catanzaro. Prefixrl: Optimization of parallel prefix circuits using deep reinforcement learning. In *Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC)*, pages 853–858. IEEE, 2021.

[8] Nan Wu, Jiwon Lee, Yuan Xie, and Cong Hao. Lostin: Logic optimization via spatio-temporal information with hybrid graph models. In *Proceedings of the 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 11–18. IEEE, 2022.

[9] Animesh Basak Chowdhury, Benjamin Tan, Ryan Carey, Tushit Jain, Ramesh Karri, and Siddharth Garg. Bulls-eye: Active few-shot learning guided logic synthesis. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. IEEE, 2022.

[10] Walter Lau Neto, Max Austin, Scott Temple, Luca Amaru, Xifan Tang, and Pierre-Emmanuel Gaillardon. Lsoracle: A logic synthesis framework driven by artificial intelligence. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–6. IEEE, 2019.

[11] Yi-Chen Lu, Wei-Ting Chan, Deyuan Guo, Sudipto Kundu, Vishal Khandelwal, and Sung Kyu Lim. Rl-ccd: Concurrent clock and data optimization using attention-based self-supervised reinforcement learning. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[12] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *arXiv preprint arXiv:1905.10947*, 2019.

[13] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[14] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *arXiv preprint arXiv:1710.10903*, 2017.

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

[16] Zhiyao Xie, Rongjian Liang, Xiaoqing Xu, Jiang Hu, Yixiao Duan, and Yiran Chen. Net2: A graph attention network method customized for pre-placement net length estimation. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 671–677. IEEE, 2021.

[17] Yi-Chen Lu and Sung Kyu Lim. On advancing physical design using graph neural networks. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7. IEEE, 2022.

[18] Christoph Albrecht. Iwls 2005 benchmarks. In *International Workshop for Logic Synthesis (IWLS)*. IEEE, 2005.

[19] Open cores. https://opencores.org/. Accessed: 2024-03-08.

[20] Synopsys Inc. Fusion compiler for simply better ppa. https://www.synopsys.com/implementation-and-signoff/physicalimplementation/fusion-compiler.html. Accessed: 2024-03-08.

[21] Yanqing Zhang, Haoxing Ren, and Brucek Khailany. Grannite: Graph neural network inference for transferable power estimation. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[22] Haoxing Ren, George F Kokai, Walker J Turner, and Ting-Sheng Ku. Paragraph: Layout parasitics and device parameter prediction using graph neural networks. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[23] Yi-Chen Lu, Siddhartha Nath, Vishal Khandelwal, and Sung Kyu Lim. Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning. In *Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC)*, pages 733–738. IEEE, 2021.

[24] Haoxing Ren, Siddhartha Nath, Yanqing Zhang, Hao Chen, and Mingjie Liu. Why are graph neural networks effective for eda problems? In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2022.

[25] Zizheng Guo, Mingjie Liu, Jiaqi Gu, Shuhan Zhang, David Z Pan, and Yibo 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)*, pages 1207–1212. IEEE, 2022.

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

[27] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *arXiv preprint arXiv:1810.00826*, 2018.

[28] Daniela Sánchez Lopera and Wolfgang Ecker. Applying gnns to timing estimation at rtl. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2022.

[29] Xiaohan Gao, Chenhui Deng, Mingjie Liu, Zhiru Zhang, David Z Pan, and Yibo Lin. Layout symmetry annotation for analog circuits with graph neural networks. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 152–157. IEEE, 2021.

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

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

[32] Anwar Said, Mudassir Shabbir, Brian Broll, Waseem Abbas, Peter Völgyesi, and Xenofon Koutsoukos. Circuit design completion using graph neural networks. In *Neural Computing and Applications*, pages 12145–12157. Springer, 2023.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems (NeurIPS)*. ACM, 2017.

[34] Siddhartha Nath, Geraldo Pradipta, Corey Hu, Tian Yang, Brucek Khailany, and Haoxing Ren. Transsizer: A novel transformer-based fast gate sizer. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–9. IEEE, 2022.

[35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *arXiv preprint arXiv:1412.6980*, 2014.

[36] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. In *arXiv preprint arXiv:1903.02428*, 2019.

[37] Scott Davidson. Characteristics of the itc'99 benchmark circuits. In *International Test Synthesis Workshop (ITSW)*. IEEE, 1999.

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

[39] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (KDD)*, pages 785–794. ACM, 2016.