GraphCAD: Leveraging Graph Neural Networks for Accuracy Prediction Handling Crosstalk-affected Delays

Fangzhou Liu
The Chinese University of Hong Kong
Hong Kong SAR

Guannan Guo
Huawei Design Automation Lab
Hong Kong SAR

Yuyang Ye
The Chinese University of Hong Kong
Hong Kong SAR

Ziyi Wang
The Chinese University of Hong Kong
Hong Kong SAR

Wenjie Fu
HiSilicon Technologies Co.
Shanghai, China

Weihua Sheng
Huawei Design Automation Lab
Hong Kong SAR

Bei Yu
The Chinese University of Hong Kong
Hong Kong SAR

Abstract

As chip fabrication technology advances, the capacitive effects between wires have become increasingly pronounced, making crosstalk-induced incremental delay a serious issue. Traditional static timing analysis involves complex and iterative calculations through timing windows, requiring precise alignment of aggressor and victim nets, along with delay and slew estimations, which significantly increase runtime and licensing costs. In our work, we develop a Graph Neural Network framework to predict crosstalk-affected delays, focusing on the impacts of the coupling effect and overlapping nets. Moreover, we employ a curriculum learning strategy that gradually integrates aggressors with victims, improving model convergence through progressively complex scenarios. Experimental results show that our framework precisely predicts crosstalk-affected delays, matching commercial tools' performance with a fivefold speedup.

CCS Concepts

• **Hardware** → **Transition-based timing analysis**; **Modeling and parameter extraction**.

Keywords

Crosstalk Analysis, Graph Learning

ACM Reference Format:
Fangzhou Liu, Guannan Guo, Yuyang Ye, Ziyi Wang, Wenjie Fu, Weihua Sheng, and Bei Yu. 2025. GraphCAD: Leveraging Graph Neural Networks for Accuracy Prediction Handling Crosstalk-affected Delays . In *Proceedings of the 2025 International Symposium on Physical Design (ISPD '25), March 16–19, 2025, Austin, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3698364.3705345

1 Introduction

As feature sizes shrink with advancing technologies, wires have been brought closer to each other. Despite reductions in wire width, heights have not proportionally decreased to avoid significant increases in resistance. This has enhanced capacitive coupling between wires, leading to increased crosstalk noise and delta delays, affecting circuit performance [1]. As illustrated in Figure 1, voltage changes in Dnet A generate currents in Dnet B through coupling capacitors, designating B as the "Victim" and A as the "Aggressor." The timing window $[T_{\min}, T_{\max}]$ represents the interval when voltage shifts are possible. If the timing windows of the victim and aggressor overlap, the aggressor can induce crosstalk in the victim.

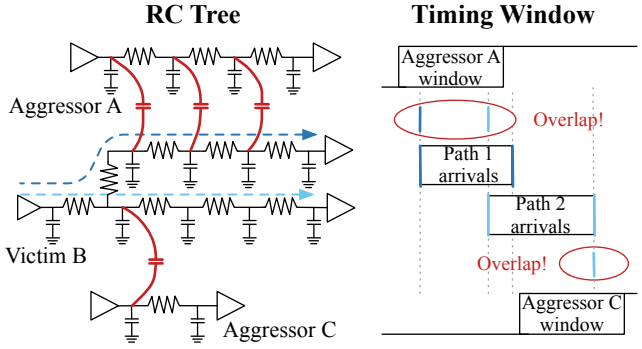


Figure 1: Primary causes of crosstalk noise.

In highly congested integrated circuits, estimating crosstalk-affected delays is crucial to precise timing closure. Simulation-based crosstalk analysis poses substantial challenges due to the complex nature of delay computations, which involve several intricate steps [2]: initially identifying and filtering minor net couplings post-routing, then calculating noise-free delays with all coupling capacitances grounded. The most crucial part is an iterative timing-window analysis, where noise from aggressor nets is calculated and added to victim nets to estimate worst-case delays. This includes: 1) Computing the noise bump for each aggressor. 2) Filtering these bumps based on peak values and logic correlation. 3) Aligning all bumps to create a combined noise waveform. 4) Overlaying this combined noise onto the noise-free waveform to determine the worst-case delay. 5) Adjusting timing windows based on these extremes for subsequent iterations.

Although classical methods utilizing iterative calculations through timing windows are accurate, recent studies have shifted towards

ISPD '25, Austin, TX, USA

ACM ISBN 979-8-4007-1293-7/25/03
https://doi.org/10.1145/3698364.3705345

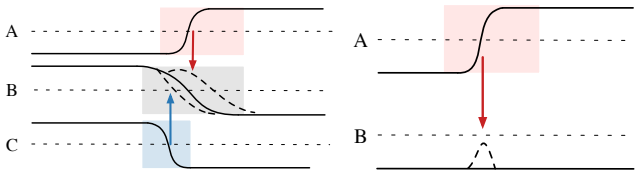
machine learning (ML) models for fast prediction. Kahng *et al.* [3] adapt support vector machine and artificial neural network techniques for crosstalk prediction. They focus on key electrical and logical structure parameters that cause timing divergence between SI and non-SI modes within commercial tools. XT-PRAGGMA [4] employs ML techniques to eliminate false aggressors and accurately predict crosstalk-affected delays. Jin *et al.* [5] introduce a two-step ML approach that predicts delay noise by integrating physically relevant and timing-specific data, such as interconnect width and net arrival times, facilitating rapid crosstalk prediction without the need for post-route netlists and parasitic data.

However, these studies do not sufficiently analyze and manage timing windows compared to classical methods, which compromise the accuracy of predictions to some extent. [6] outlines two critical issues affecting the analysis of timing windows, which inform the design considerations for our work:

(1) **Coupling effect**: Parasitic cross-coupling between adjacent nets can cause transition slowdowns or speedups (known as delta delays) or voltage glitches (known as bumps), triggered by charge transfers that cause voltage fluctuations. The left part of Figure 1 displays a model of cross-coupled nets, featuring a victim net and two aggressor nets. The crosstalk from the aggressor nets, transmitted through four coupling capacitors, impacts the victim net. Note that the coupling capacitors and RC values already account for the distance between the nets.
(2) **Overlapping net**: The right part of Figure 1 analyzes the timing windows of each net, where a delta delay occurs only when the arrival windows of the aggressor and victim paths overlap. Commercial tools utilize several methods to assess overlap relationships. The example in Figure 1 uses the "all path edges" mode to independently evaluate the victim wire path's early or late arrival times. For maximum-delay analysis, the victim net is impacted by aggressor nets A and C due to overlapping late edges on at least one path. Similarly, for minimum delay analysis, the victim is affected solely by aggressor C, excluding the influence of aggressor A based on the overlap.

In addressing crosstalk analysis challenges, we are inspired by wire timing prediction research such as [7]. This research motivates us to use graph neural networks (GNNs) to accurately predict delays caused by crosstalk by effectively handling coupling capacitor features and timing windows, thus avoiding classical time-consuming and costly computations or simulations. Additionally, given the complexity of analyzing all potential aggressors on a victim during overlapping net analysis, we innovatively propose a curriculum learning approach as discussed in [8]. This method initially targets the most significant aggressors and incrementally includes less impactful ones, thereby optimizing model training by systematically escalating complexity. Building on this, we introduce the **GraphCAD** framework, summarizing its key contributions as follows:

- An end-to-end framework is developed to predict crosstalk-affected delays, considering coupling effects and overlapping net analysis.
- We introduce a customized GNN that applies heterogeneous graph learning for analyzing the impact of coupling capacitance on crosstalk, and uses a graph transformer model to capture key features and map out relationships among wire paths.



(a) delay noise (b) function noise

Figure 2: Crosstalk delay effects. Note that greater timing window overlap indicates stronger crosstalk.

- We apply a curriculum learning strategy that targets dynamics between victim nets and multiple aggressor nets, smoothly progressing the model from simple to complex graph challenges.
- Experimental studies conducted on extensive open-source designs at the 7nm node validate our framework's efficiency and accuracy in predicting crosstalk effects.

2 Preliminaries

2.1 Crosstalk Analysis

Crosstalk significantly impacts signal integrity (SI) as it involves unwanted electrical interactions between two or more physically adjacent nets due to capacitive cross-coupling. In this context, a net affected by such interference is termed a "victim net," while a net causing these effects is known as an "aggressor net." As depicted in Figure 2(a), voltage fluctuations in aggressor A/C can influence victim B's voltage, altering the waveform and introducing delay variations known as delay noise. Moreover, Figure 2(b) illustrates that when victim B's voltage is stable, changes in the aggressor's voltage can cause a bump in the victim's circuit. If this bump is sufficiently strong, it might alter voltage levels in the victim's circuit, potentially causing execution errors known as function noise. In this paper, we exclusively focus on delay noise, as the calculations for function noise typically depend on the timing window determined by delay noise calculations.

In classical algorithms, the calculation of delta delay considers the worst-case impact of aggressors on a victim. We introduce the concept of a timing window to identify which aggressors alter voltage simultaneously with the victim and influence the delta delay calculation. Notably, the computation of the timing window also depends on the delta delay itself. For instance, if the timing window for the current victim's input voltage is $[T_{\min}, T_{\max}]$, and the net's delay is $[\text{delay}_{\min}, \text{delay}_{\max}]$, the next stage's input voltage timing window is calculated as $[T_{\min} + \text{delay}_{\min}, T_{\max} + \text{delay}_{\max}]$. Thus, the calculations of delta delay and the timing window are iterative, starting with an infinite timing window and considering noise from all neighboring aggressors. Each iteration adjusts and excludes some aggressors based on the previous timing window, gradually narrowing it until convergence. Determining the worst-case aggressors within this window involves a complex nonlinear optimization, usually solved approximately. First, calculate the noise (bump) each aggressor produces when acting alone on a fixed voltage victim, then sum all bumps to obtain the total noise. Next, this total noise is superimposed on the victim's noise-free waveform to analyze the worst-case noise overlay effect, thereby approximating the delta delay caused by aggressors in the worst case. Note that the likelihood of crosstalk between the victim net and aggressor

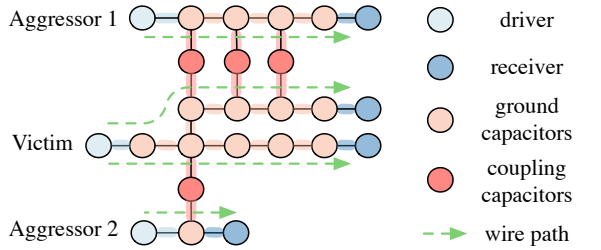


Figure 3: Illustration of RC-VA graph. Drivers, receivers, and capacitances serve as nodes; resistances serve as edges.

nets increases with greater overlap in their timing windows, due to intensified signal interactions.

2.2 Graph Neural Network

Graph neural networks have emerged as a powerful framework for graph data analysis, widely recognized for their iterative message-passing scheme [9]. In a GNN, each node u in a graph $G = \langle \mathcal{V}, \mathcal{E} \rangle$ updates its embedding $\boldsymbol{h}_u^{(k)}$ iteratively. The update during each pass is based on messages aggregated from the node's neighbors $\mathcal{N}(u)$:

$$\begin{aligned} \boldsymbol{h}_u^{(k+1)} &= \text{UPDATE}^{(k)}\left(\boldsymbol{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\boldsymbol{h}_v^{(k)}, \forall v \in \mathcal{N}(u))\right) \\ &= \text{UPDATE}^{(k)}\left(\boldsymbol{h}_u^{(k)}, \boldsymbol{m}_{\mathcal{N}(u)}^{(k)}\right), \end{aligned} \tag{1}$$

where UPDATE and AGGREGATE are differentiable functions and $\boldsymbol{m}_{\mathcal{N}(u)}$ means the "message" aggregated from u's graph neighborhood $\mathcal{N}(u)$. After K iterations, the final node embedding for all nodes u in $\mathcal{V}$ is $\boldsymbol{z}_u = \boldsymbol{h}_u^K$.

This approach has shown significant efficacy in learning complex graph structures and has recently become popular in the EDA community, as circuits can be effectively modeled as graphs [10].

2.3 Problem Formulation

We aim to create a fast and accurate framework for estimating crosstalk-affected delays in complex RC net structures.

Problem 1 (Crosstalk-Affected Timing Estimation)**.** Given complex RC nets with victim and aggressor configurations, this problem seeks to assess their coupling effects and net overlaps to accurately forecast wire delay and slew at the victim net's sink pin.

3 Algorithm

3.1 Overview

Before diving into the algorithm details, we begin by giving an overview of our proposed GraphCAD framework, depicted in Figure 4. Our approach consists of three primary modules:

- A heterogeneous graph attention network (HGAT) model for modeling coupling effects;
- A graph transformer network (GraphTransformer) model for wire path feature and overlapping net analysis;
- A multi-head neural network that utilizes a multilayer Perceptron (MLP) to integrate embeddings and perform diverse objective readouts.

Starting from the RC tree extracted from the routed circuit, we identify victim-aggressor pairs and construct their corresponding graph. The HGAT model uses a bi-level attention mechanism to deeply explore the heterogeneity of RC trees, focusing specifically

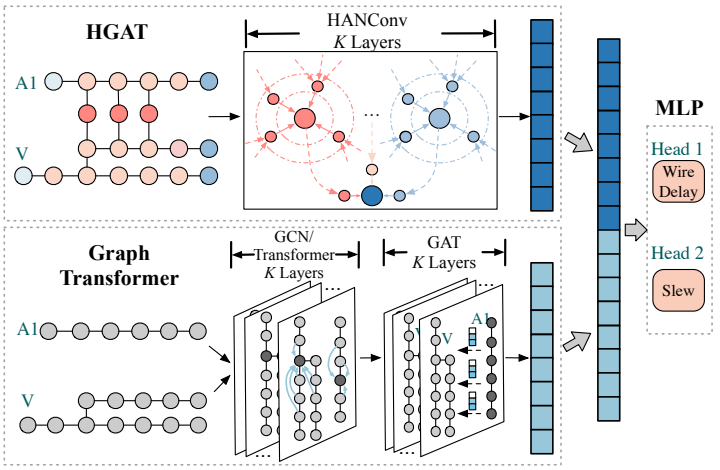


Figure 4: Overview of GraphCAD.

on coupler information. The Graph Transformer model employs GraphSAGE to extract local features by aggregating edge-based information and uses a Transformer to integrate global structural data, capturing all capacitance and resistance within the RC net. It further incorporates GAT layers to specifically analyze wire path interactions, especially how aggressor wire paths impact victim wire path timing windows. The outputs of the HGAT and Graph-Transformer models are then concatenated and processed through MLP layers to predict wire delay and sink pin's slew.

3.2 Data Representation

Graph Construction. The RC tree is derived from the standard parasitic exchange format (SPEF) file after routing. Subsequently, we identify pairs of victim and aggressor nets, and construct a heterogeneous graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, as illustrated in Figure 3. This graph is termed the RC-VA graph, where the node set $\mathcal{V}$ encompasses drivers, receivers, ground capacitors, and coupling capacitors, while its edge set $\mathcal{E}$ consists of resistors that connect these various node types. In this graph, the term "driver" refers to the source pin of each net, whereas "receiver" denotes the sink pin. Additionally, a "wire path" is defined as the route from a driver to a receiver within a single net, characterized by the number of receivers it includes.

For different model training, which is detailed in Section 3.3, the graph $\mathcal{G}$ is initially transformed into two distinct data structures. For the input to the heterogeneous graph attention network (HGAT) model, the heterogeneous graph is treated as a whole and represented by a node feature dictionary $\boldsymbol{X}_{\text{dict}}$ and an edge index dictionary $\boldsymbol{E}_{\text{dict}}$. $\boldsymbol{X}_{\text{dict}}$ includes node features for each node type, $\{\text{"coup"} : \boldsymbol{x}_{\text{coup}}, \text{"cap"} : \boldsymbol{x}_{\text{cap}}, \ldots\}$, and the $\boldsymbol{E}_{\text{dict}}$ contains local adjacency information for each edge type, $\{(\text{"cap"}, \text{"coup"}) : \boldsymbol{e}_{\text{cap, coup}}, \ldots\}$. Conversely, for the input to the Graph Transformer model, the heterogeneous graph $\mathcal{G}$ is decoupled and split into multiple isomorphic subgraphs for both victim and aggressor nets, represented as $\mathcal{G} = (\mathcal{E}, \mathcal{V}, \mathcal{P})$ where $\mathcal{P}$ denotes wire paths. Each subgraph treats nodes and edges uniformly without differentiation by type and is described by a node feature matrix $\boldsymbol{X}$, a path feature matrix $\boldsymbol{P}$ for $\mathcal{P}$ and a weighted adjacency matrix $\boldsymbol{A} = [a_{i,j}]$, with $a_{i,j}$ indicating the resistance between nodes v_i and v_j.

Table 1: Description of node and path features.

Feature	Description
f_{n1}	Capacitance values
f_{n2}	Number of input nodes
f_{n3}	Number of output nodes
f_{n4}	Total input capacitance
f_{n5}	Total output capacitance
f_{n6}	Number of connected resistors
f_{n7}	Total input resistance
f_{n8}	Total output resistance
f_{n9}	Ratio of coupling-to-total capacitance
f_{n10}	Indicates if it is a victim net
f_{n11}	List of corresponding aggressors
f_{p1}	Incremental delay for each wire path
f_{p2}	Minimum transition time for driver/receiver
f_{p3}	Maximum transition time for driver/receiver
f_{p4}	Minimum arrival time for driver/receiver
f_{p5}	Maximum arrival time for driver/receiver

Feature Extraction. Let $\boldsymbol{f}_n = (f_{n1}, f_{n2}, \ldots, f_{n11})$ represent the node features obtained from the SPEF file, and $\boldsymbol{f}_p = (f_{p1}, f_{p2}, \ldots, f_{p5})$ represent the path features sourced from an industrial standard timer in Non-SI mode. These features are listed in Table 1.

Given that negative coupling increases delay while positive coupling decreases it, our experiments are specifically designed to focus on a primary scenario. We examine cases where the rise time of signals in the victim net aligns with the fall time of signals in the aggressor nets, implicitly addressing its inverse as well. Additionally, we utilize the composite current source (CCS) model and its accompanying technology library for feature extraction. This model effectively handles non-linear effects and various switching conditions, including signal rise and fall times, which makes the previously suggested Elmore model unnecessary, as referenced in [7]. Our method specifically addresses the delta delay caused by crosstalk between the driver and receivers, intentionally omitting unrelated characteristics like cell load strength from the technology library. This focused approach enhances our analysis of crosstalk effects and their impact on signal integrity, minimizing the interference from process variations.

We predict the total wire delay and receiver's slew affected by crosstalk using neural network models:

$$t_{\text{wire}} = g(\boldsymbol{f}_n, \boldsymbol{f}_p; \theta_g), \tag{2}$$

$$t_{\text{trans}} = h(\boldsymbol{f}_n, \boldsymbol{f}_p; \theta_h), \tag{3}$$

where g and h are functions modeled by neural networks, parameterized by θ_g and θ_h respectively. By analyzing non-linear dependencies between node features ($\boldsymbol{f}_n$) and path features ($\boldsymbol{f}_p$), these models aim to isolate and quantify changes, thereby accurately predicting the crosstalk's impact on both wire delays and receiver's slew.

Ground truth. To simulate crosstalk effects accurately, we perform dynamic SPICE simulations, modeling each wire path from driver to receiver. This allows precise measurements of wire delay $\boldsymbol{d}_g$ and receiver's slew $\boldsymbol{t}_g$. Aligning with our feature extraction, we focus solely on the victim net at maximum rise and the aggressor at maximum fall.

3.3 GraphCAD Framework Details

HGAT Model: Coupling Effect Analysis. As mentioned before, we model the RC tree as heterogeneous graphs with diverse node types, where neighboring nodes with different types indicate distinct relationships. Observing that different node types and relationships contribute unequally to the coupling effects, we propose to employ the heterogeneous graph attention (HGAT) model [11] to encode our RC-VA graph for coupling effect analysis effectively.

We begin with the formal definition of relation:

Definition 1 (Relation)**.** A relation Θ is defined as a pair of node types in the form of $t_i \rightarrow t_j$, which describes a composite relation between connected objects n_1 of type t_i and n_2 of type t_j.

Our HGAT model follows the two-level message aggregation scheme that first encodes intra-relation information and then aggregates relation-level information. Initially, we leverage self-attention [12] to learn the weight among neighboring nodes of the same type. Given a node pair (u, v) which is connected via relationship Θ, the node-level attention e^{Θ} representing how important node v is for node u can learn the importance $e^{\Theta}_{u,v}$ as:

$$e^{\Theta}_{u,v} = \sigma(a^{\top}_{\Theta} \cdot [h_i || h_j]). \tag{4}$$

Here h_u is the embedding of node u, σ is the activation function, a_{Θ} is a learnable attention vector for relation Θ, and $||$ denotes the concatenate operation. Notably, a_{Θ} is shared for all node pairs of relation Θ. To ensure comparability of these importance scores across different nodes, we normalize them using softmax:

$$\alpha^{\Theta}_{u,v} = \text{softmax}(e^{\Theta}_{u,v}) = \frac{\exp(e^{\Theta}_{u,v})}{\sum_{k \in \mathcal{N}^{\Theta}(u)} \exp(e_{u,k})}, \tag{5}$$

where $\mathcal{N}^{\Theta}(v)$ is the set of neighboring nodes of v related by Θ.

Then the relation-based embedding of node u can be produced by the weighted linear combination of the neighboring nodes of relation Θ, which can be written as:

$$z^{\Theta}_{u} = \sigma\Big(\sum_{v \in \mathcal{N}^{\Theta}(u)} \alpha^{\Theta}_{u,v} h_v\Big), \tag{6}$$

where z^{Θ}_{u} represents the embedding of node u for relation Θ. Given the relation set $\{\Theta_1, \cdots, \Theta_K\}$, the node-level aggregation scheme produces K groups of relation-level node embeddings, denoted as $\{Z_{\Theta_1}, \cdots, Z_{\Theta_K}\}$.

To generate more comprehensive node embeddings, we further apply an additional attention mechanism to learn the importance of different relation-level embeddings and fuse them into a single embedding. To begin with, we first transform relation-level embeddings through a nonlinear transformation, e.g., an MLP layer. Following this, the importance of a relation-level embedding is measured as the similarity of the transformed embedding with a semantic-level attention vector q. Notably, q is shared for all nodes and relations. To represent the importance of each relation Θ_i, we average the importance of all the relation-level node embeddings belonging to Θ_i, which can be written as follows:

$$e_{\Theta_i} = \frac{1}{|\mathcal{V}_{\Theta_i}|} \sum_{u \in \mathcal{V}_{\Theta_i}} \boldsymbol{q}^{\top} \cdot tanh(W \times z^{\Theta_i}_{u} + b), \tag{7}$$

where W is the weight matrix, b is the bias vector, q is the relation-level attention vector, and $\mathcal{V}_{\Theta_i}$ represents the set of nodes belonging to relation Θ_i. Similarly, the above relation-level importance scores are normalized using the softmax function:

$$\alpha_{\Theta_i} = \text{softmax}(e_{\Theta_i}) = \frac{\exp(e_{\Theta_i})}{\sum_{j=1}^{K} \exp(e_{\Theta_j})}, \tag{8}$$

where K denotes the number of relations. With the above normalized importance scores, we can fuse the relation-level node embeddings to generate the final embeddings as follows:

$$Z = \sum_{i=1}^{K} \alpha_{\Theta_i} \times Z_{\Theta_i}. \tag{9}$$

After propagation through the `HANConv` layers, a global pooling layer is applied to the final node embeddings to summarize information across the graph:

$$\begin{aligned} \boldsymbol{y}_{\text{HGAT}} &= GlobalPool(Z) \\ &= [(\frac{1}{|\mathcal{V}|} \sum_{u \in \mathcal{V}} z_u) || (\frac{1}{|\mathcal{V}^{coup}|} \sum_{v \in \mathcal{V}^{coup}} f(x_v))], \end{aligned} \tag{10}$$

where z_u denotes the generated node embedding for node u, $\mathcal{V}^{coup}$ represents the set of coupler nodes, || denotes the concatenate operation, and f is an MLP deep learning model. Notably, we add a skip connection to emphasize the pivotal role of coupler nodes.

Graph Transformer: Overlapping Net Analysis. Despite the HGAT model's focus on utilizing diverse node features, it falls short in leveraging timing path information. Inspired by [7], we introduce a Graph Transformer model that combines GraphSAGE and Transformer modules to efficiently capture local and global structures from wire paths. We then use a pooling module to integrate these paths' representations and establish virtual connections based on temporal overlaps. These connections, indicating overlapping nets, are processed using graph attention network (GAT) layers to simulate path interactions. Finally, we aggregate the enhanced features of the victim net to produce the final aggregated feature.

The process begins by applying GraphSAGE convolutions to node features of decomposed isomorphic graphs representing victim and aggressor wire paths. Recall that for this model's input, coupler nodes are removed, and each wire path is processed independently. This process specifically focuses on learning local structures by aggregating features from neighboring nodes, based on the edge weights $\boldsymbol{A} = [a_{i,j}]$ (the resistance values between nodes), to capture key connectivity patterns. Each node's feature vector is updated through L_1 layers of GraphSAGE as follows:

$$\boldsymbol{x}_v^{(l+1)} = \text{ReLU}\left(\text{Norm}\left(\boldsymbol{W}^{(l)} \cdot \text{MEAN}\left(\{\boldsymbol{x}_v^{(l)}\} \cup \{\boldsymbol{x}_u^{(l)} : u \in \mathcal{N}(v)\}\right)\right)\right), \tag{11}$$

where $\boldsymbol{x}_v^{(l)}$ denotes the feature vector of node v at layer l ($l < L_1$), and $\mathcal{N}(v)$ denotes the set of neighboring nodes of v in the graph. Note that for the first layer, $\boldsymbol{x}_v^{(0)}$ is the original node feature $\boldsymbol{x}_v \in \boldsymbol{X}$ defined in Section 3.2.

As we know that GraphSAGE suffers from performance degradation due to oversmoothing when increasing model depth [13], and its narrow receptive field limits capturing global dependencies. To address these issues, we integrate a multi-head self-attention mechanism from transformers after the GraphSAGE layers. Our model does not use explicit position embeddings but learns to infer positional relationships implicitly during the learning process:

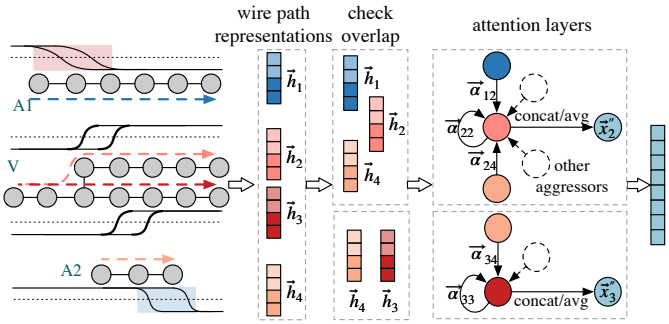


Figure 5: Graph Transformer model incorporating overlapping net information.

$$\boldsymbol{x}'_v = \text{TransformerEncoder}(\boldsymbol{x}_v^{(L_1)}, L_2), \tag{12}$$

where $\boldsymbol{x}_v^{(L_1)}$ denotes the node embeddings from the final GraphSAGE layer, and $\boldsymbol{x}'_v$ is the output after processing through L_2 transformer layers, enhancing the model's capability to capture both local and global dependencies.

Subsequently, we utilize a pooling module to integrate node representations $\boldsymbol{X}^{L_1+L_2} = \{\boldsymbol{x}_i^{(L_1+L_2)} \mid i \in \mathcal{V}\}$ with wire path features $\boldsymbol{P} = \{\boldsymbol{p}_i \mid i \in \mathcal{P}\}$, forming wire path representations $\boldsymbol{H} = \{\boldsymbol{h}_i \mid i \in \mathcal{P}\}$. As shown in Figure 5, after obtaining $\boldsymbol{H}$, we perform an overlapping check between victim and aggressor wire paths, adhering to the "all-path-edges" concept mentioned in [6]. We establish the timing edges based on the minimum and maximum arrival times at each wire path's receiver, which are saved in $\boldsymbol{P}$. The timing window on the receiver is considered because it reflects the total duration for the signal to traverse from the driver through the entire net to the receiver's endpoint. In minimum delay analysis, if the early edge t_{early} of a victim wire path overlaps with the timing window Δt_{agg} of any aggressor, i.e., $t_{\text{early}} \cap \Delta t_{\text{agg}} \neq \emptyset$, the paths are considered correlated. Similarly, in maximum delay analysis, overlap between the late edge t_{late} and Δt_{agg} also indicates a correlation, i.e., $t_{\text{late}} \cap \Delta t_{\text{agg}} \neq \emptyset$. Thus, the correlation between a victim and an aggressor net is established if either condition is met, expressed as $(t_{\text{early}} \cap \Delta t_{\text{agg}}) \cup (t_{\text{late}} \cap \Delta t_{\text{agg}}) \neq \emptyset$. We denote the set of all correlated pairs as $\mathcal{C}$, which includes any pair $(v, \mathcal{A}_v)$ where v is a victim wire path and $\mathcal{A}_v$ contains all aggressor wire paths correlated with v. To allow for a more flexible determination of overlap and to mitigate close but non-overlapping edges, we empirically introduce extra padding to the timing edges.

Recall that in the Graph Transformer input stage, aggressor and victim wire paths are considered unconnected; therefore, for those crosstalking victim and aggressor wire paths in $\mathcal{C}$, we introduce a virtual edge. Subsequently, we apply GAT layers to calculate weighted features between each victim wire path and all its crosstalking aggressor wire paths in $\mathcal{C}$, resulting in a new wire path representation enriched with critical information. Thus, even if two wire paths are not directly connected, the features of an unconnected aggressor can influence the victim through attention-weighted updates.

$$\boldsymbol{x}''_v = \sum_{u \in \mathcal{N}_{\mathcal{C}}(v)} \alpha_{vu} \boldsymbol{W} \boldsymbol{x}_u, \tag{13}$$

where $\mathcal{N}_{\mathcal{C}}(v)$ denotes the set of aggressor paths for victim v, α_{vu} are the attention coefficients, and $\mathbf{W}$ is a learnable weight matrix.

After processing, features for each pair $(v, \mathcal{A}_v)$ in $\mathcal{C}$ are concatenated and averaged to produce the final model output $\boldsymbol{y}_{\text{GTrans}}$, which is the mean of concatenated features $\{\boldsymbol{x}''_{v,a} : (v, \mathcal{A}_v) \in \mathcal{C}\}$. This output vector $\boldsymbol{y}_{\text{GTrans}}$ summarizes key features and interactions for crosstalk-affected delay predictions.

Multi-Head Prediction. Finally, the multi-head prediction model uses outputs from the HGAT and Graph Transformer models to predict crosstalk-affected wire delays and receiver's slew through dedicated prediction heads. The model includes:

- Combined Feature Representation:

$$\boldsymbol{y}_{\text{combined}} = [\boldsymbol{y}_{\text{HGAT}} \parallel \boldsymbol{y}_{\text{GTrans}}], \tag{14}$$

  where $\parallel$ denotes the concatenation operation.
- Prediction Heads and Regularization:

$$\boldsymbol{d}_e = \text{Linear}(\boldsymbol{y}_{\text{combined}}) \rightarrow \text{LeakyReLU} \rightarrow \text{Dropout} \rightarrow \text{Linear}, \tag{15}$$

$$\boldsymbol{t}_e = \text{Linear}(\boldsymbol{y}_{\text{combined}}) \rightarrow \text{LeakyReLU} \rightarrow \text{Dropout} \rightarrow \text{Linear}. \tag{16}$$

The final outputs, $\boldsymbol{d}_e$ and $\boldsymbol{t}_e$, quantify crosstalk-affected delays, leveraging the concatenated features for enhanced model accuracy.

3.4 Curriculum Learning Mechanism

Crosstalk analysis is a complex, iterative process that involves examining the interactions between a victim net and its aggressor nets. The impact of the aggressor nets on the victim net can alter the timing performance of the victim net, which, in turn, influences the aggressor nets. To address the above challenge, we propose to employ the curriculum learning mechanism [8] during our training stage. Given an RC tree, we initially concentrate solely on the victim net and gradually incorporate the impact of aggressor nets based on their criticality. Here we define the criticality of an aggressor net based on the "all-path-edge" concept [6] as its degree of correlation with the victim net, specifically measured by the number of overlaps between the aggressor's timing window and the victim's timing edges. As shown in Figure 6, A_1 is deemed more critical than A_2 because it has two timing window overlaps with the victim's timing edges, compared to only one for A_2. Building on this approach, our training flow boosts performance by prioritizing critical aggressor nets while incrementally increasing the complexity.

During training, as data is input, we progressively decompose each RC-VA graph—initially focusing on the victim alone, then adding the primary aggressor, followed by the secondary aggressor, and so forth. See Figure 6 (right) for a visual representation:

- **Phase 1:** Start with the victim net alone. (C_1)
- **Phase 2:** Introduce the most critical aggressor net. (C_2)
- **Phase 3:** Incrementally add more aggressor nets based on their criticality. (C_3)

Datasets are fed into the GraphCAD framework for training sequentially from C_1 to C_3. Each intermediate subgraph generated during this phased approach offers distinct insights, enhancing the training dataset. By leveraging detailed subgraph information, the model effectively handles complexities and improves its generalization across the entire RC-VA graph.

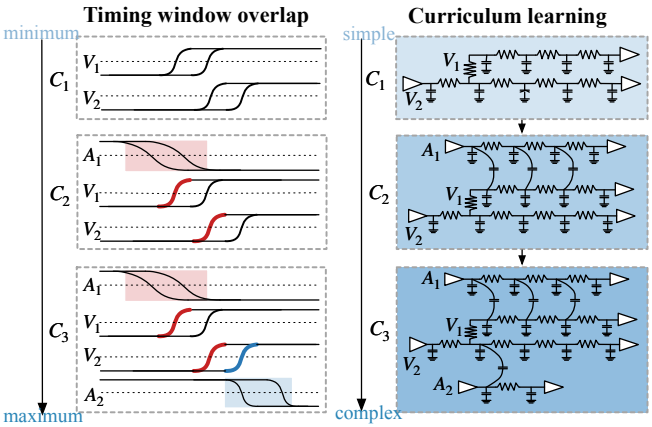


Figure 6: Illustration of the curriculum learning mechanism.

To maximize the benefits of curriculum learning, we further design a customized loss function that can be written as follows:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} C \cdot \mathcal{L}(y_i, f(x_i; \theta)), \tag{17}$$

where θ represents the model parameters and $\mathcal{L}$ is the mean squared error (MSE) between the predicted and actual target delays. N indicates the total number of samples, with y_i and x_i as the target and input values for the i-th sample, respectively. The complexity coefficient C, defined as the count of aggressor networks plus one, weights the MSE of each sample to scale the loss with scenario complexity. This approach ensures that more complex scenarios exert a greater influence on model tuning. Additionally, the optimizer's learning rate is dynamically adjusted, decreasing gradually to manage complexity and fine-tune parameters, thus preventing overshooting in complex environments.

Note that in the final inference stage, we only need to input a complete RC-VA graph to obtain the predicted results. This curriculum learning strategy customizes training to meet the specific challenges posed by the timing window, ensuring dynamic adaptation to the cumulative effects of multiple aggressors.

4 Experiments

4.1 Dataset Preparation

We source our datasets from Opencores [14] and other open-source hardware projects, as detailed in Table 2. The open-source 7nm process design kit, ASAP7 [15], equipped with a CCS timing model, is utilized to accurately characterize transistor delays. Our workflow incorporates commercial tools for synthesis, placement, and routing, followed by the collection of post-route Verilog and SPEF files. To enhance model training, we apply a filtering step to identify victim-aggressor pairs with significant crosstalk effects in the dataset. This filtering is performed using an industrial-standard timer, referred to as "Timer," in SI-mode, which has proven successful in sign-off analysis. We extract pairs exhibiting pronounced crosstalk using the `report_si_bottleneck` command. Note that this filtering does not affect the precision of our inference results, ensuring that predictions remain as accurate as if the data were unfiltered.

For relative feature extraction, we employ Timer in Non-SI mode to generate timing reports. From these, we extract critical timing

Table 2: IP cores in training, validation, and test sets.

Net Range	IP	
	Train/Valid	Test
<1w	ps2 [14], uart16550 [14], cavlc [14], wb_dma [14], gng [14], ac97 [14], mem_ctrl [14], mc6809 [14], darkriscv [14], ha1588 [14], tv80 [14], oc8051 [14], yacc [14]	wdsp [14], ae18 [14]
<10w	picorv32 [14], WinoGen [19], pci [14], arm9 [14], aes_core [14], LoopBranchPredictor [20], sha3 [14], mips32r1 [14], aes [14], ecg [14], ethmac [14], CSR [20], yadmc [14], LoadQueue [21]	wb2axip [14], LSU [20], fpu [14], vga_lcd [14], usb_device [14]
<100w	L2TLB [20], rvsteel [14], fft256 [14], MediumBoom [20], Sha3Rocket [22], Aquarius [14], MediumOctoBoom [20], MegaOctoBoom [20], Cache [21], LargeBoomAndRocket [22]	SmallBoom [20], BoomCore [20], or1200 [14], SmallQuadBoom [20], sparc [14]

information for specific nets of interest, such as the arrival times of drivers and receivers, which are detailed in Table 1.

To gather ground truth data, we employ Timer to generate the spice decks for commercial SPICE simulations. Each victim net and its corresponding aggressor nets are automatically assigned simulation parameters. The aggressor nets are prioritized in the simulation settings file based on the extent of their crosstalk impact, facilitating the collection of subgraphs for curriculum learning.

4.2 Experimental Setup

We develop the framework with PYG [16] and PyTorch [17]. The GraphCAD framework is trained and evaluated on a machine with 2.40GHz CPUs and two GPUs (10752 cores, 625.0 GFLOPS for FP64, 24GB of main memory). We initially partition the dataset into training, evaluation, and testing subsets to ensure generalizability. During data loading, we utilize a SPEF parser [18] to extract RC parasitics and then construct RC-VA graphs along with associated features.

The training process aims to minimize the customized loss function detailed in Equation (17), comparing estimated wire delay and receiver's slew ($\boldsymbol{d}_e$, $\boldsymbol{t}_e$) against the ground truth ($\boldsymbol{d}_g$, $\boldsymbol{t}_g$). Our model features a five-layer `HANConv` with 32 hidden dimensions and 4 heads for the HGAT model. The Graph Transformer model includes 32 hidden channels with a dual-layer architecture of 5 layers each for GraphSAGE and Transformer modules, supported by 8-head attention and 0.2 dropouts. The multi-head prediction model combines outputs from HGAT and Graph Transformer, utilizing a hidden layer of size 128 for predicting delay features. We train our model with a dynamically adjusted learning rate starting from 0.01 to 0.006 and a batch size of 128 over 150 epochs. The training process takes approximately 20 hours on a single GPU. Our approach relies solely on net information and learned parameters, allowing the inductive model to be applied to different designs without compromising accuracy, even on unseen data.

4.3 Overall Performance

We compare our method with the commercial Timer and previous state-of-the-art (SOTA) works for predicting crosstalk-affected delays. Timer employs a traditional crosstalk analysis workflow that includes electrical filtering and iterative delay calculation, often resulting in pessimistic worst-case timing estimates. We also select the SOTA NetTiming [7] for comparison, which uses GNNTrans to estimate post-routing wire timing. However, this work does not adequately address SI-analysis-related features, such as coupling effect analysis. Additionally, we explore SI-driven timing prediction studies like those in [3, 5]. Since these works have not been open-sourced, we incorporate only the relevant features described, along with the MLP mentioned in their studies, for our comparison. Table 3 provides detailed statistics of the benchmark for the test dataset. The overall performance comparison is presented in Table 4, and the corresponding runtime data can be found in Table 5.

Table 3: Test dataset statistics: g _cap (ground), c_cap (coupling), x_cap (extraction) capacitors.

Benchmarks	#Inst	#Nets	#FF	g_cap	c_cap	x_cap
wdsp	5570	4850	890	3546	12778	1533
ae18	6942	6730	841	4930	33043	6924
wb2axip	15976	14885	1899	36102	43545	14901
usb_device	18115	18047	3756	20284	45966	7614
fpu	40079	36129	6433	85948	115629	35362
LSU	45413	43373	8757	185958	160272	68444
vga_lcd	85041	84560	17050	308980	241027	86749
SmallQuadBoom	170092	163276	20267	595202	584103	223029
SmallBoom	170287	163867	20189	643150	554004	241167
BoomCore	183703	178480	25438	788018	622525	295780
or1200	475718	656852	110553	507880	1319171	209013
sparc	856180	920686	179320	1165820	2504447	467219

In Table 4, we present the effective RC-VA graphs used in our experiments, where victim wire paths are extracted from these graphs. The ratio of estimation error to victim wire paths for each benchmark is expressed as $\frac{EE}{N_{vnp}}$, where EE denotes the estimation error and N_{vnp} is the total number of victim wire paths. To further elucidate, the estimation error is derived by comparing each work's calculated or predicted results with the SPICE simulation results, where the estimation error for SPICE is inherently zero. In the comparison of wire delays, Timer demonstrates the highest accuracy, with GraphCAD's estimation error being 8.21% higher than Timer's but still lower than the other two baselines, NetTiming and MLP, by 5.40% and 17.73%, respectively. Regarding the slew for receivers, GraphCAD outdoes Timer by 2.74% and significantly surpasses the others by 10.46% and 26.79%. This considerable edge over other machine learning-based methods in predicting delays highlights the effectiveness of the GraphCAD framework. Although our proposed framework does not outperform Timer in both two metrics, it is noteworthy that GraphCAD operates in less than 25% of the runtime required by Timer. This suggests that our approach is practically valuable and efficient. Moreover, as Figure 7 shows, we analyze the runtime speedup of our method against our baseline and SPICE. We find that SPICE is notably slower in simulating large batches of nets, with some simulations exceeding 10 seconds. All machine learning (ML)-based approaches have reduced runtime by more than tenfold, with our work achieving a 33.16% speedup. As anticipated, ML-based prediction models are faster than traditional methods, reaffirming the need to develop rapid and effective techniques to assess delays caused by crosstalk.

Table 4: Comparison of estimation errors against SPICE results.

Benchmarks	RC-VA	Predicted Wire Delay				Predicted Slew at Receiver			
		Timer	NetTiming [7]	MLP	GraphCAD	Timer	NetTiming [7]	MLP	GraphCAD
wdsp	19	**14.89%**	31.32%	37.37%	25.66%	47.36%	30.04%	80.80%	**26.03%**
ae18	11	**13.27%**	23.39%	24.34%	16.42%	35.95%	35.71%	72.96%	**34.82%**
wb2axip	22	14.45%	24.37%	22.14%	**12.51%**	17.26%	22.54%	33.75%	**10.17%**
usb_device	99	**14.91%**	22.38%	31.21%	21.18%	30.46%	28.37%	53.25%	**27.93%**
fpu	183	**10.19%**	23.00%	29.05%	20.25%	21.82%	34.64%	39.79%	**20.84%**
LSU	31	**7.93%**	22.83%	41.00%	18.31%	39.30%	40.03%	80.16%	**36.81%**
vga_lcd	24	**15.82%**	32.18%	65.24%	19.59%	43.70%	24.33%	97.15%	**20.45%**
SmallQuadBoom	511	**6.62%**	26.43%	27.71%	24.07%	**16.72%**	36.18%	29.63%	22.28%
SmallBoom	402	**8.44%**	25.64%	34.83%	21.92%	**20.26%**	35.08%	29.75%	23.52%
BoomCore	326	**11.41%**	31.92%	33.50%	24.91%	**16.84%**	45.77%	25.57%	24.05%
or1200	2	12.36%	14.15%	76.56%	**9.42%**	**15.86%**	47.58%	42.14%	21.79%
sparc	174	**10.30%**	26.30%	29.00%	24.93%	**19.47%**	37.44%	28.72%	23.48%
Average	-	**11.72%**	25.33%	37.66%	19.93%	27.08%	34.81%	51.14%	**24.35%**
Delta	-	**-8.21%**	5.40%	17.73%	0	2.74%	10.46%	26.79%	**0**

Table 5: Runtime comparison.

Benchmarks	Runtime (s)				
	SPICE	Timer	NetTiming [7]	MLP	GraphCAD
wdsp	34.222	12.363	3.004	2.641	2.126
ae18	18.687	11.780	3.252	2.945	1.559
wb2axip	38.667	15.045	4.297	1.180	2.188
usb_device	169.281	16.026	4.838	1.225	5.905
fpu	321.501	21.878	5.862	1.199	9.246
LSU	55.532	24.155	5.296	1.117	2.371
vga_lcd	46.886	16.284	5.896	1.131	2.222
SmallQuadBoom	903.420	43.565	7.397	2.886	25.861
SmallBoom	709.360	44.576	7.516	1.242	20.769
BoomCore	590.145	67.807	7.204	2.812	20.486
or1200	3.649	37.710	2.639	2.554	0.906
sparc	535.247	112.374	6.390	2.687	9.716
Average	285.550	35.297	5.299	1.968	8.613
Ratio	33.154	4.098	0.615	0.229	1.000

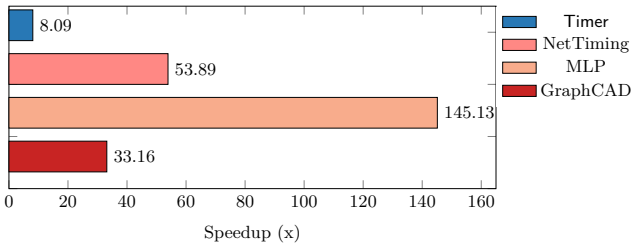


Figure 7: Illustration of runtime speedup.

4.4 Curve Estimation & Fit

Crosstalk analysis often leads to noise analysis, prompting us to extend our prediction framework to include the transition waveform at the receiver. We derive our ground truth data from `.tr0` files generated by SPICE and focus on capturing the dynamics of transition times by selecting 50 key points between 0 and VDD. Our analysis concentrates on the x-coordinates while keeping the y-coordinates fixed. To improve curve estimation, we've enhanced the output channels of our multi-head prediction model, replacing the MLP delay head with LSTM layers to better capture the continuity of the x-coordinates. During our evaluation phase, the mean error rate, calculated as the average of normalized absolute differences between the predicted and target points (x-values), is below 10% in 15.7% of all test cases. This suggests that, while there is clearly room for improvement, GraphCAD still provides a reliable reference point for ongoing optimizations. As depicted in Figure 8, we highlight an instance of curve prediction where we achieve a mean error rate of approximately 1.24%.

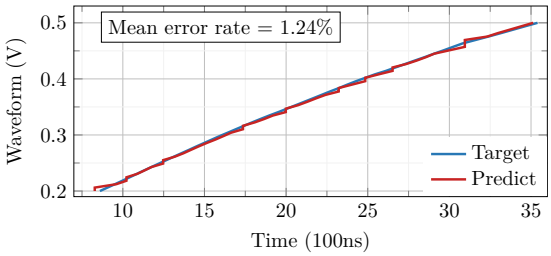


Figure 8: Curve prediction for SmallQuadBoom/n76718. Waveform shows receiver U64214/B2's transition.

5 Conclusion

In summary, this work presents the GraphCAD framework, a prediction model for crosstalk-affected delays that highlights the significance of coupling effects and overlapping nets in crosstalk analysis. We innovatively apply a curriculum learning strategy, incorporating both the HGAT model and the graph transformer model with customized features, to achieve precise predictions. Experimental results from real-world designs demonstrate that our GraphCAD framework performs the prediction tasks both accurately and efficiently, outperforming commercial tools and SOTA methods. Future efforts will concentrate on leveraging the predicted data for noise analysis and optimization tasks.

Acknowledgments

This work is partially supported by The Research Grants Council of Hong Kong SAR (No. CUHK14210723 and No. CUHK14211824), and the MIND project (MINDXZ202404).

References

[1] S. Sapatnekar, *Timing*. Springer Science & Business Media, 2004.

[2] S. Sirichotiyakul, D. Blaauw, C. Oh, R. Levy, V. Zolotov, and J. Zuo, "Driver modeling and alignment for worst-case delay noise," in *ACM/IEEE Design Automation Conference (DAC)*, 2001, pp. 720–725.

[3] A. B. Kahng, M. Luo, and S. Nath, "SI for free: machine learning of interconnect coupling delay and transition effects," in *ACM Workshop on System Level Interconnect Prediction (SLIP)*, 2015, pp. 1–8.

[4] V. A. Chhabria, B. Keller, Y. Zhang, S. Vollala, S. Pratty, H. Ren, and B. Khailany, "XT-PRAGGMA: Crosstalk pessimism reduction achieved with GPU gate-level simulations and machine learning," in *ACM/IEEE Workshop on Machine Learning CAD (MLCAD)*, 2022, pp. 63–69.

[5] L. Jin, J. Xu, W. Fu, H. Yan, and L. Shi, "A crosstalk-aware timing prediction method in routing," *arXiv preprint arXiv:2403.04145*, 2024.

[6] Synopsys, *Primetime User Guide*, 2019.

[7] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi, "Fast and accurate wire timing estimation based on graph learning," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. IEEE, 2023, pp. 1–6.

[8] X. Wang, Y. Chen, and W. Zhu, "A survey on curriculum learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 44, no. 9, pp. 4555–4576, 2021.

[9] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artifical Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

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

[11] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The Web Conference*, 2019, pp. 2022–2032.

[12] A. Vaswani, "Attention is all you need," *Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[13] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *International Conference on Machine Learning (ICML)*. PMLR, 2020, pp. 1725–1735.

[14] "Opencores hardware rtl designs," https://opencores.org.

[15] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm FinFET Predictive Process Design Kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[16] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[18] "Parser-spef," https://github.com/OpenTimer/Parser-SPEF.

[19] M. Li, P. Li, S. Yin, S. Chen, B. Li, C. Tong, J. Yang, T. Chen, and B. Yu, "WinoGen: A Highly Configurable Winograd Convolution IP Generator for Efficient CNN Acceleration on FPGA," in *ACM/IEEE Design Automation Conference (DAC)*, 2024.

[20] K. Asanovic, D. A. Patterson, and C. Celio, "The Berkeley Out-of-order Machine (BOOM): An Industry-competitive, Synthesizable, Parameterized RISC-V Processor," 2015.

[21] Y. Xu, Z. Yu, and D. e. a. Tang, "Towards Developing High Performance RISC-V Processors Using Agile Methodology," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 1178–1199.

[22] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.