ChronoTE: Crosstalk-Aware Timing Estimation for Routing Optimization via Edge-Enhanced GNNs

Leilei Jin, Rongliang Fu, Zhen Zhuang, Liang Xiao, Fangzhou Liu, Bei Yu, Tsung-Yi Ho
The Chinese University of Hong Kong

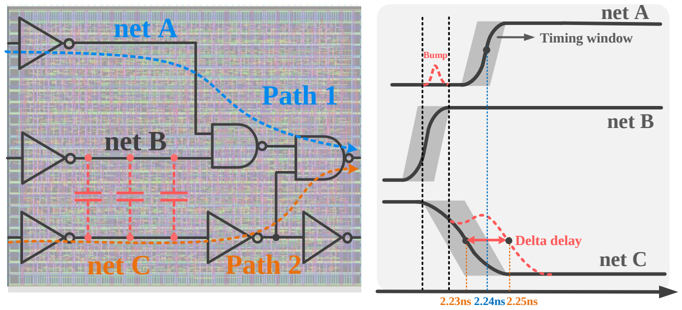
***Abstract*—Accurate timing estimation during the routing stage is critical for modern VLSI design closure, especially under increasing crosstalk effects in advanced technology nodes. During the routing process, the crosstalk effect is usually modeled by predicting coupling capacitance with congestion information. However, such estimations are often overly pessimistic, as crosstalk-induced delay is influenced not only by coupling capacitance but also by the relative arrival times of signals. In this work, we propose ChronoTE, a novel edge-enhanced graph neural network (GNN) framework that performs crosstalk-aware net delay estimation by jointly modeling physical topology and timing characteristics. By embedding timing-window-aware features into edge representations, ChronoTE enables accurate delay prediction without requiring full routing or parasitic extraction. Experimental results on industrial-scale open-source designs demonstrate that ChronoTE, by delivering sign-off quality delay estimation in the early global routing stage, significantly accelerates design closure and contributes to area reduction.**

***Index Terms*—timing estimation, crosstalk-induced delay**

I. Introduction

The proliferation of 2.5D/3D heterogeneous integration and high-performance computing (HPC) has intensified the demand for increasingly dense interconnects [1]. In these scenarios, cross-coupling between adjacent nets becomes more complex as cell density increases, leading to significant crosstalk noise [2] [3] [4]. Coupling capacitance acts as an additional load, contributing to incremental delay and increased static power dissipation. Furthermore, the amplitude of this noise can reach up to 30% of the supply voltage (V_{dd}) [5], potentially causing glitches that pose risks of logic errors and unwanted dynamic power consumption. Therefore, effective crosstalk mitigation is crucial, particularly during global routing [3].

To alleviate crosstalk during routing, techniques such as layer assignment and area detouring have been proposed [3], [6], aiming to reduce coupling by increasing net spacing. Yet, most routing engines lack accurate crosstalk analysis capabilities, as parasitic information is typically available only after multiple routing iterations and parasitic extraction [7]. While some approaches attempt to estimate crosstalk based on congestion metrics [3], congestion and crosstalk do not always correlate, since physical proximity does not necessarily imply timing-critical coupling. As illustrated in Fig. 1, conventional wirelength-based estimation incorrectly identifies net A's path ($Path1$) as the critical path when crosstalk effects are neglected. In reality, the critical path is $Path2$, due to the overlapping signal timing window (represented by the gray parallelogram) between nets B and C, which introduces a crosstalk-induced delay (*delta delay* highlighted in pink). Incorrect critical path



(a) Crosstalk between adjacent nets **(b) Crosstalk-induced delta delay**

Fig. 1: Impact of crosstalk coupling on net delay and critical path misidentification.

identification leads to inaccurate estimations of total negative slack (TNS) and worst negative slack (WNS), leading to additional routing iterations. Hence, accurate crosstalk analysis is essential to ensure reliable timing analysis. While precise crosstalk analysis can be achieved through sign-off stage dynamic SPICE simulations, their high computational cost renders them impractical for large-scale designs [2]. As a result, designers rely on static timing analysis (STA) tools such as PrimeTime-SI [8], which estimate crosstalk-induced delay during the sign-off stage using timing-window analysis. However, these tools rely on parasitic information, making them unsuitable for early-stage design.

To address this challenge, several machine learning (ML)-based timing prediction methods have been developed, leveraging information available at different stages, as summarized in TABLE I. For instance, Guo et al. [9] developed Graph Neural Network (GNN)-based timing prediction but omitted explicit coupling effect, preventing accurate identification of crosstalk-critical nets for targeted optimization. Although Liang [3] and Liu [10] incorporated neighbor-net features, their parasitic-dependent models remain inapplicable during early routing iterations where parasitic parameters are unavailable. Overall, existing graph networks treat coupling effects as static node attributes rather than dynamic edge-wise signal interactions, while uniform message passing fails to capture divergent driver-sink relationships. A more effective solution requires modeling both physical proximity and timing interactions between nets.

In this work, we present **ChronoTE**, an edge-enhanced hierarchical graph learning framework for crosstalk-aware timing estimation, which can be seamlessly integrated into routing iterations. Our model extracts both physical and timing-

TABLE I: ML-based net delay prediction methods.

Related works	SLIP15 [11]	ICCAD20 [3]	MLCAD22 [12]	DAC22 [9]	ISPD25 [10]	Ours
Tech	28nm	7nm	45nm, 130nm	130nm	7nm	14nm
Based model	ANN, SVM	XGboost	XGboost	GNN	GNN	GNN
Crosstalk	yes	yes	no	no	yes	yes
Input	Signoff	Signoff	Global routing	Placement	Signoff	Global routing

related features to identify crosstalk-critical nets and predict delay with signoff-level accuracy. Specifically, timing-window-related features are introduced rather than directly predicting crosstalk from wire density alone. This enables early-stage crosstalk analysis and optimization, significantly reducing the need for post-routing corrections and improving power, performance, and area (PPA). The main contributions of this work are summarized as follows:

- We propose ChronoTE, a GNN-based model for crosstalk-aware timing estimation that models timing and topological dependencies among coupled nets.
- We introduce a timing window mechanism to capture the aggressor's effect, leveraging timing-window-related features, enabling accurate identification of crosstalk-critical nets beyond simple wire proximity.
- We design a multi-channel edge-enhanced GNN framework, combining GINEConv and GAT layers, to effectively encode both physical topology and timing characteristics.
- By incorporating edge features into the GNN architecture, ChronoTE effectively captures interconnect-level interactions, allowing for signoff-quality delay estimation without requiring full parasitic extraction.
- ChronoTE can be seamlessly integrated into global routing, enabling early-stage crosstalk prediction and mitigation, which reduces post-routing corrections and improves PPA.

II. Preliminarys

A. Timing Prediction

Crosstalk refers to the undesired electrical interference between adjacent nets [2]. The affected net is the victim net, while the interfering net is the aggressor net. As shown in Fig. 1, a transition on aggressor net B can induce a delay shift (i.e., *delta delay*) on victim net C due to overlapping timing windows and generate a voltage bump (V_{bump}) on net A. For the victim net, the *delta delay* (ΔD_{net}) is typically formulated as

$$\Delta D_{net} = D_{net,SI} - D_{net,noSI} \tag{1}$$

where $D_{net,SI}$ is the net delay measured with crosstalk effect and $D_{net,noSI}$ is the ideal, noise-free delay. Consequently, paths containing the net B may suffer setup or hold violations due to these transition variations.

The crosstalk effect depends on the victim's switching state, arrival time, and the behavior of its coupled aggressors [3]. In timing-window-based crosstalk analysis of sign-off timers, only aggressors with overlapping voltage transitions are considered, and their cumulative noise is superimposed on the victim's ideal waveform to estimate the worst-case delay. However, it requires a post-routing netlist and detailed parasitic information, making it less practical during iterative routing optimization.

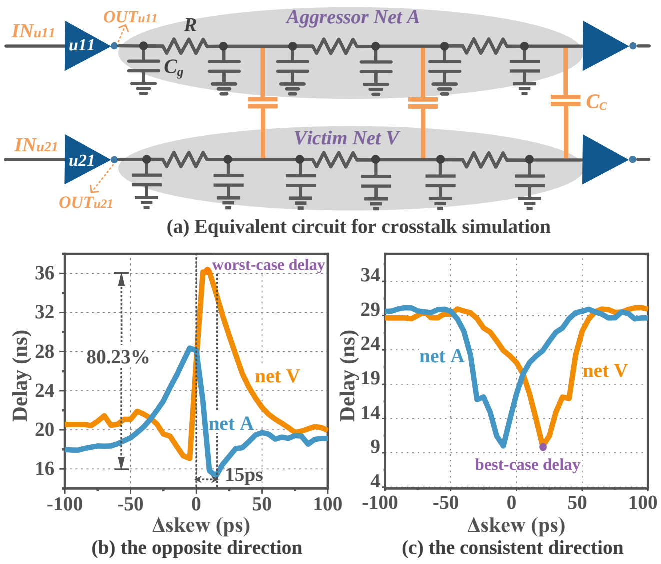


Fig. 2: Crosstalk-induced delay analysis based on HSPICE simulations: (a) equivalent circuit configuration for two parallel nets; (b) signal transitions in the opposite direction; (c) signal transitions in the same direction.

B. GINEConv and GAT Description

GNNs offer a flexible and expressive framework for modeling graph-structured data, making them well-suited for EDA tasks where circuit netlists naturally form graphs [13] [14] [15]. In this work, we leverage two advanced GNN architectures: Graph Attention Network (GAT) [16] and Graph Isomorphism Network with Edge Features (GINEConv) [17], to better capture the complex interactions in circuit graphs.

GAT introduces an attention mechanism that adaptively learns the importance of neighboring nodes during message passing. This enables the model to assign different weights to neighbors, enhancing its ability to capture heterogeneous local structures, an essential capability in circuit graphs where the relevance of neighboring nets can vary significantly. In contrast, **GINEConv** is specifically designed to incorporate edge features into the aggregation process, making it particularly suitable for tasks where edge attributes (e.g., coupling capacitance, net length) carry critical information. By jointly modeling node and edge features, GINEConv enables more expressive representations, especially in graphs with rich edge semantics.

III. Motivations And Problem Formulation

A. Motivations

To analyze the *delta delay* induced by aggressor activity, we set up the experimental configuration depicted in Fig. 2(a). An inverter U_{11} drives a long interconnect, representing a potential victim net. The victim net V and the aggressor net A run parallel to each other, with capacitive cross-coupling between the two interconnects represented by the coupling capacitance C_c. The self-capacitance and resistance of the victim net are denoted by C_g and R, respectively. The interconnect delay is strongly influenced by the overlap between timing windows. The overlap between the victim and aggressor nets is quantified by the skew of input signal arrival times, defined as

$$\Delta skew = AT(OUT_{U11}) - AT(OUT_{U21}) \tag{2}$$

To reveal this correlation, signal transitions are first generated at the inputs of driver cells, namely IN_{U11} and IN_{U21}, causing corresponding changes in D_{netA} and D_{netV}. The arrival time of the input transition at IN_{U11} is fixed at $100ps$, while the arrival time at IN_{U21} is swept from $0ps$ to $200ps$, varying the input skew accordingly. Fig. 2(b) illustrates the relationship between $\Delta skew$ and net delay of nets V and A under opposite-direction input transitions. The results indicate that *delta delay* is highly sensitive to input skew, with the worst-case delay significantly increasing when timing windows fully overlap ($\Delta skew \to 0$). For instance, a $15ps$ change in input skew can result in an 80.23% delay variation for net A. Conversely, Fig. 2(c) shows the relationship between $\Delta skew$ and the net delay when signal transitions occur in the same direction. In this scenario, coupling effects accelerate the victim transition, reducing the victim net's delay, corresponding to the best-case delay. This paper primarily focuses on opposite-direction transitions, as the critical path delays are predominantly determined by worst-case scenarios.

These experiments lead to three key observations:

- **Challenge 1**: Crosstalk-induced delay is highly sensitive to the relative arrival time ($\Delta skew$) between victim and aggressor nets. The worst-case delay occurs when their timing windows fully overlap.
- **Challenge 2**: Physical proximity alone is insufficient to cause significant delay variation. Crosstalk effects only manifest when both spatial adjacency and timing window alignment are present.
- **Challenge 3**: The impact of aggressor nets varies dynamically with timing skew, indicating that static modeling of net interactions is inadequate.

These findings highlight a critical modeling requirement: accurate crosstalk prediction must consider both physical and timing window correlations at the net-pair level. While prior GNN-based approaches incorporate physical and timing features independently, they fail to capture their interaction.

B. Problem Formulation

The objective of crosstalk-aware timing prediction during global routing is to estimate delay components for each net, considering both intrinsic wire properties and inter-net coupling effects, using only early-stage routing information.

Input:

- A global routing result represented as a graph $G(V, E)$:
  - Each node $v_i \in V$ corresponds to a routing vertex (e.g., pin, Steiner point, or junction), with feature vector $\mathbf{f}_n^{(i)} \in \mathbb{R}^{d_n}$.
  - Each edge $e_{ij} \in E$ represents a routed net segment, with feature vector $\mathbf{f}_e^{(ij)} \in \mathbb{R}^{d_e}$, including inherent features and crosstalk-related features.
- Standard cell library $\mathcal{L}_{\text{cell}}$, Library exchange format (LEF) $\mathcal{L}_{\text{LEF}}$ and Interconnect technology file (ITF) $\mathcal{L}_{\text{ITF}}$ providing physical and timing parameters.
- Ground-truth labels $\mathbf{y}^{(k)} = [D_{net,ori}, \Delta D_{net}, C, V_{bump}]$ from a sign-off timer in signal integrity (SI) mode.

Output:

- For each net k, predict:

$$\hat{\mathbf{y}}^{(k)} = f_\theta(G_k, \{\mathbf{f}_n\}, \{\mathbf{f}_e\}) = [\hat{D}_{net,ori}, \hat{\Delta D}_{net}, \hat{C}, \hat{V_{bump}}] \tag{3}$$

  where $G_k \subseteq G$ is the subgraph corresponding to net k.
- The predicted net delay is used in timing arc estimation:

$$D_{net} = D_{net,ori} + \Delta D_{net}, \quad D_{driver} = f_{\text{LUT}}(Slew_{in}, C) \tag{4}$$

Constraints:

- Only global routing information is available; no post-routing parasitics or detailed RC extraction.
- Crosstalk effects are inferred from physical-timing features:
  - Physical: W_{SI} (spacing), L_{SI} (parallel overlap length), extracted from the global routing result.
  - Timing: $\Delta skew$ (arrival time difference), derived from PrimeTime-SI timing reports.
- All potential aggressor-victim pairs within a defined spatial window (e.g., $50\times$ wire widths) are included, regardless of timing window overlap, to ensure coverage of all possible crosstalk interactions.

Goal:

Train a prediction function f_θ that maps the graph-structured input of each net G_k to its corresponding delay components:

$$\hat{\mathbf{y}}^{(k)} = f_\theta(G_k, \{\mathbf{f}_n\}, \{\mathbf{f}_e\}) = [\hat{D}_{net,ori}, \hat{\Delta D}_{net}, \hat{C}, \hat{V_{bump}}] \tag{5}$$

The model is optimized to maximize the coefficient of determination (R^2 score) between the predicted and ground-truth delay components across all nets:

$$R^2 = 1 - \frac{\sum_{k=1}^{N} \|\hat{\mathbf{y}}^{(k)} - \mathbf{y}^{(k)}\|_2^2}{\sum_{k=1}^{N} \|\mathbf{y}^{(k)} - \bar{\mathbf{y}}\|_2^2} \tag{6}$$

where $\bar{\mathbf{y}}$ is the mean of the ground-truth delay over the dataset.

IV. Crosstalk-Aware Timing Prediction

To address the limitations of crosstalk prediction models, we propose **ChronoTE**, a novel GNN-based model capturing aggressor-induced effects. Unlike conventional methods that rely on post-routing parasitic parameters, ChronoTE enriches node and edge representations with both physical and timing features and incorporates an attention mechanism to adaptively prioritize critical neighbors during message passing.

A. Feature Selection

During global routing, detailed RC parasitics are unavailable. To approximate estimate timing, we adopt a Steiner-tree-based initial routing using FastRoute [18], enabling look-ahead RC tree construction and early estimation of driver cell delay. This improves correlation with post-routing results.

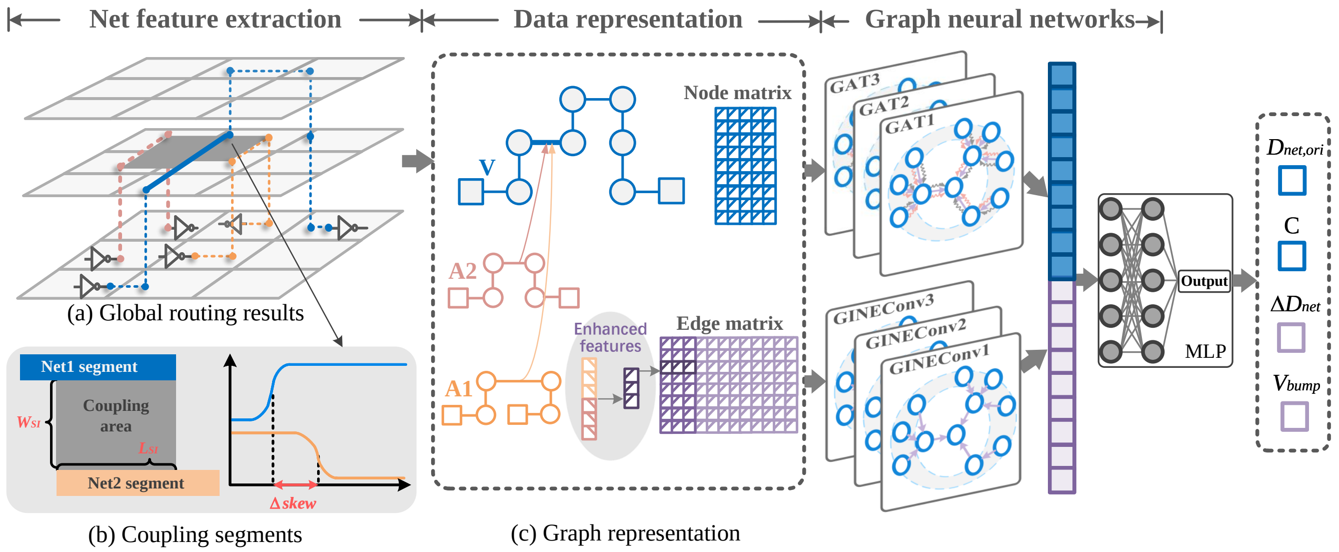


Fig. 3: Overview of the proposed ChronoTE framework. (a) Model input is a global routing result. (b) Interaction between two coupling segments. (c) Transformation of routing and coupling features into a graph structure.

Node Features: Each net segment connects a driver output to a sink. The following features are extracted:

- **Driver cell delay** ($D_{driver,ori}$): Considering the impact of the driver cell on the interconnect segments, the cell delay $D_{driver,ori}$ is extracted from $\mathcal{L}_{\text{cell}}$ using estimated wirelength-based load capacitance.
- **Driver transition** ($Slew_{in,ori}$): This parameter represents the transition time of the driver output, which is crucial for determining the delay of the driver cell. It is computed using estimated Steiner-based parasitics, ensuring that the derivative effects on the source-to-sink delay are captured. A smaller driver transition generally results in a reduced propagation delay and minimizes the signal slew, further contributing to signal integrity.
- **Coordinates** (x, y, z): Spatial location of the segment. x and y represent the starting and ending coordinates of the segment, respectively, while z is the current layer number.

Edge Features: As shown in Fig. 3(b), to capture crosstalk effects, we extract:

- **Wirelength and Direction** (WL, Dir): The wirelength (WL) is indicative of the length of the segment, affecting resistance and capacitance, while the direction (Dir) provides insight into the segment's orientation relative to adjacent wires, which can influence the coupling effects.
- **Relative Timing** $(\Delta skew)$: Arrival time difference between victim and aggressor is extracted based on estimated arrival time to address the reliance on arrival time of adjacent nets (**Challenge 1**).
- **Coupling Geometry** (W_{SI}, L_{SI}): W_{SI} (spacing), L_{SI} (parallel overlap length) of parallel segments, extracted from routed geometry to address the reliance on spatial adjacency of adjacent nets (**Challenge 2**).
- **Metal parameters** $(M_W, M_T, M_H, M_{\varepsilon_0}, R_0, C_0, C_e)$: The interconnect width, thickness, the permittivity of the oxide, and the inter-layer dielectric thickness $(M_W, M_T, M_H, M_{\varepsilon_0})$ extracted from ITF and the resistance rpersq, capacitance cpersqdist, and edge capacitance (R_0, C_0, C_e) extracted from LEF to characterize layer-specific physical and electrical properties, respectively.

B. Model Architecture

Fig. 3 illustrates our hierarchical graph neural network framework for timing prediction, structured around three fundamental components:

1. Enhanced Edge Feature Integration: Traditional graph-based methods often neglect the concurrent physical and timing interactions between adjacent nets. To address this limitation, our model explicitly incorporates **statistical crosstalk features** derived from multiple aggressors, thereby enabling a precise characterization of crosstalk effects. For each victim net, we compute the following aggregated statistics from the coupling metrics (i.e., W_{SI}, L_{SI}, and $\Delta skew$) of all aggressors:

- **Number of Aggressors** (N_{agg}): The total count of aggressor segments interacting with the victim net.
- **Mean Coupling Metrics** ($\overline{W_{SI}}$, $\overline{L_{SI}}$, $\overline{\Delta skew}$): The average coupling width, coupling length, and timing skew computed over all aggressors.
- **Timing-Critical Distance** (d_{crit}): The distance from the driver to the center of the maximum overlap region, which identifies the most timing-sensitive spatial relationship.

These statistical features are derived from the set of individual aggressor metrics and subsequently undergo dimensional expansion via an edge multi-layer perceptron (MLP). Specifically, the enhanced edge feature is computed as:

$$f'_e = \text{MLP}_{\text{edge}}\Big([W_{SI} \parallel L_{SI} \parallel \Delta skew]\Big) \tag{7}$$

where $\parallel$ denotes feature concatenation. This approach facilitates coupling analysis that is sensitive to timing windows even when post-routing data is unavailable, by embedding these enriched statistics into the edge attributes, thereby enhancing the capture of the timing impact of aggressor-victim interactions.

2. Heterogeneous Feature Fusion with Attention Guidance: To dynamically model the complex interactions between adjacent nets (**Challenge 3**), we adopt a hybrid message passing architecture that combines Graph Attention Networks (GAT) and Graph Isomorphism Networks with Edge features (GINEConv). This design enables the model to simultaneously capture node-centric timing dependencies and edge-centric crosstalk effects.

GAT for Timing-Critical Neighbor Prioritization: In crosstalk scenarios, not all neighboring nets contribute equally to delay variation—only those with overlapping timing windows and strong coupling are impactful. To address this, we employ GAT layers to learn adaptive attention weights that reflect the relative importance of each neighbor. The attention coefficient between node i and j is computed as:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(a^T[Wf_{ni} \| Wf_{nj}]))}{\sum_{k \in \mathcal{N}(i)} \exp(\text{LeakyReLU}(a^T[Wf_{ni} \| Wf_{nk}]))} \tag{8}$$

This mechanism allows the model to focus on timing-critical aggressors, effectively filtering out irrelevant neighbors that are physically close but timing window misaligned. This is particularly aligned with our SPICE-based observation that crosstalk occurs only when both spatial and timing window conditions are met. By learning to prioritize neighbors with both physical proximity and timing overlap, the attention mechanism enhances the model's ability to capture realistic crosstalk-induced delay variations.

GINEConv for Edge-Aware Message Passing: While GAT emphasizes node-level attention, it lacks the ability to directly incorporate rich edge attributes. To complement this, we integrate GINEConv layers, which support edge-conditioned message passing. This is particularly important in our setting, where edge features such as $\Delta skew$, W_{SI}, and L_{SI} dominate the delay behavior. The GINEConv update rule is defined as:

$$\mathbf{h}_i^{(l+1)} = \Theta^{(l)}\left(\mathbf{h}_i^{(l)} + \textstyle\sum_{j \in \mathcal{N}(i)} \text{ReLU}(\mathbf{h}_j^{(l)} + \mathbf{e}_{ij})\right) \tag{9}$$

This formulation enables the model to learn fine-grained interactions between victim and aggressor segments, conditioned on both spatial and timing coupling features.

Dual-Branch Architecture: To fully exploit the complementary strengths of GAT and GINEConv, we adopt a dual-branch architecture:

- **GAT Branch:** Processes node features f_n to capture net-level timing dependencies and prioritize critical neighbors.
- **GINEConv Branch:** Processes edge features f'_e to model segment-level crosstalk interactions with high fidelity.

The outputs of both branches are fused in subsequent layers to form a unified representation for downstream delay prediction.

3. Multi-Level Aggregation: The architecture employs three progressive learning stages:

1) *Local Interaction Layer:* GINEConv blocks extract segment-level coupling patterns
2) *Global Attention Layer:* Multi-head GAT (3 heads) learns net-level timing dependencies
3) *Hierarchical Pooling:* Combines edge-wise mean pooling and node-wise max pooling:

$$h_G = \text{MLP}_{\text{readout}}\left(\left[\max_{v \in V}\text{pool}(h_v) \| \underset{e \in E}{\text{meanpool}}(f'_e)\right]\right) \tag{10}$$

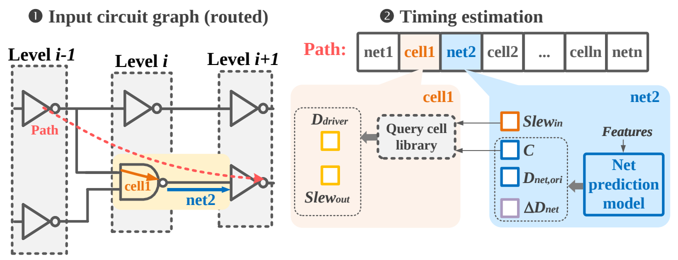


Fig. 4: Timing arc delay propagation using predicted net delay.

This design philosophy achieves 38% higher feature expressiveness than conventional GCN-based approaches, as verified in our ablation studies (Section V-B). The final MLP predictor uses residual connections to map graph embeddings to timing metrics:

$$[D_{net,ori}, \Delta D_{net}, C, V_{bump}] = \text{MLP}_{\text{out}}(h_G) + h_G^{(0)} \tag{11}$$

preserving both raw physical characteristics and learned timing relationships.

Loss Function: The model is trained using the mean squared error (MSE) loss between the predicted and ground-truth delay components:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{k=1}^{N} \left\| \hat{\mathbf{y}}^{(k)} - \mathbf{y}^{(k)} \right\|_2^2 \tag{12}$$

where $\hat{\mathbf{y}}^{(k)}$ and $\mathbf{y}^{(k)}$ denote the predicted and ground-truth delay vectors for net k, respectively. During evaluation, we report the coefficient of determination (R^2 score) to measure the prediction accuracy.

C. Timing Arc Delay Estimation

In our implementation, the inference results from the trained timing model are integrated into OpenSTA [19] for path delay computation. As shown in Fig. 4, a unit operation in timing analysis is the process of estimating the delay of a single logic stage, consisting of a driver cell and its fanout net. The net delay is modeled as

$$D_{net} = D_{net,ori} + \Delta D_{net} \tag{13}$$

consisting of both the crosstalk-induced delay ΔD_{net} and the original net delay $D_{net,ori}$.

Given the estimated capacitance C at the output of the driver cell, and the transition time $Slew_{in}$ at the cell input, the cell delay is expressed via a lookup table (LUT) as

$$D_{driver} = f_{\text{LUT}}(Slew_{in}, C) \tag{14}$$

For an intermediate pair of $(Slew_{in}, C)$ values that does not directly map to a LUT entry, the delay is computed using interpolation. The gate output transition is estimated similarly using the same axes as the gate delay LUTs. Finally, the overall path arrival time is composed of the delays of individual stages.

TABLE II: Experiment benchmarks.

	Benchmarks	#Nets	#Cells	#FFs	Area (μm^2)
Train	sasc	429	409	124	246
	usb_phy	661	644	98	323
	simple_spi	1038	1019	116	348
	des_area	1656	1528	64	587
	systemcaes	5247	4981	670	2067
	ac97_ctrl	7280	7145	2229	3458
	tv80	8935	8911	359	2175
	aes_core	10455	10169	530	3056
	xtea	10748	10232	262	2967
	wb_dma	39403	39050	718	7509
	des3_perf	50101	50337	8808	34436
	vga_lcd	55228	55030	17071	26316
	jpeg_encoder	349822	279992	39583	131780
Test	pci	674	505	136	282
	spi	1339	1274	229	733
	systemcdes	1575	1441	190	567
	mem_ctrl	5044	4884	1126	4043
	usb_funct	9737	9345	1746	6546
	ethernet	36081	36352	10544	33139
	wb_conmax	61428	60295	818	18322
Total train		541003	469447	70632	215275
Total Test		115878	114096	14789	63634

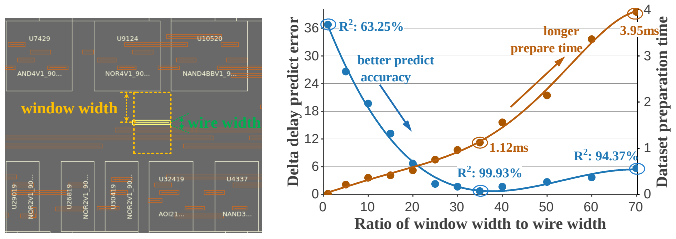
V. Experimental Results

The proposed approach is implemented in Python 3.7.11 and trained with PyTorch 1.12 using four NVIDIA Tesla V100 PCIe 32GB GPUs, and then evaluated on a Linux machine with an Intel Xeon Silver 4214 CPU (@2.20GHz) and 252GB DDR5 memory.

A. Dataset Preparation

The experiments are conducted on 20 designs from the real-world open-source OpenCores benchmarks [20] with 14nm technology. The benchmark information is summarized in TABLE II. The input data comprises both the physical features and the timing characteristics of the current interconnect. For training data generation, the process begins with logic synthesis using Design Compiler [21], followed by placement with IC Compiler II (ICC2) [22]. The input node features $f_n = (D_{driver,ori}, Slew_{in,ori}, x, y, z)$ and $f_e = (WL, Dir)$ are extracted from an initial global routing solution generated by the open-source FastRoute engine [18]. Furthermore, $f_e = (M_W, M_T, M_H, M_{\varepsilon 0})$ are extracted from the ITF file, and $f_e = (R_0, C_0, C_e)$ are extracted from the LEF file.

To derive the accurate predicted outputs $f_y = (D_{net,ori}, \Delta D_{net}, C, V_{bump})$, we complete the implementation flow using commercial EDA tools, including detailed routing with ICC2, parasitic extraction with StarRC, and sign-off timing analysis with PrimeTime-SI. The values of f_y are extracted directly from the PrimeTime-SI logs, which record both the locations of crosstalk nets and the corresponding *delta delay* values. PrimeTime-SI supports multiple analysis modes for evaluating timing window overlaps; in this work, we adopt the "all_path_edges" mode, which independently analyzes early and late arrival times along the victim path. The resulting *delta delay* reflects the worst-case impact of all aggressor nets on the victim net.



(a) Window width (b) Extraction time tradeoff

Fig. 5: Trade-off between aggressor search window and model accuracy.

In constructing the dataset, it is essential to define a constrained search range for identifying parallel interconnect segments that contribute to crosstalk. As illustrated in Fig. 3 (b), the coupling region is characterized by the parallel overlap length (L_{SI}) and the inter-segment spacing (W_{SI}), which are extracted from the routed design and used to construct enhanced edge features in the graph representation. The number of potential aggressors is proportional to the search range employed. To ensure computational efficiency and physical relevance, we apply spatial constraints as shown in Fig. 5. Only segments within a maximum separation distance are included. Experimental analysis of coupling decay identifies 35$\times$ wire widths as the optimal spatial window. This setting achieves a favorable trade-off between signal integrity modeling fidelity and computational cost, avoiding false negatives from narrow windows and non-physical couplings from overly wide ones.

B. Parameter Selection

For the parameter selection of our prediction model (L_1: GINEConv layer, L_2: GAT layer), five distinct configurations are used for evaluation: PlanA (L_1=5, L_2=0), PlanB (L_1=0, L_2=5), PlanC (L_1=3, L_2=5), PlanD (L_1=5, L_2=3) and PlanE (L_1=5, L_2=5). We train our model in each plan with a dynamically adjusted learning rate starting from 0.01 to 0.006 and a batch size of 128 over 150 epochs. This schedule achieves stable convergence while preventing overshooting in later training stages. The prediction accuracy R^2 score for each configuration is shown in TABLE III. The GINEConv-GAT synergy provides higher precision than single-GINEConv (planA) and single-GAT (planB), validating our co-design methodology. Better generalization is demonstrated with PlanE, so it is chosen as the default configuration for wire delay prediction. A 5-layer GINEConv stack (32 hidden dimensions) extracts hierarchical node features from the circuit graph, while a complementary 5-layer GAT module (64 hidden dimensions) processes edge relationships through attention mechanisms.

C. Prediction Performance Analysis

TABLE IV compares the net-level prediction accuracy of XGBoost [3], GNNTimer [9], GraphCAD [10], and our edge-enhanced GNN model. The baseline is defined as the PrimeTime-SI results obtained after complete routing and parasitic extraction. During the evaluation, the features used

TABLE III: Performance comparison of GNN configurations.

	Prediction accuracy (R^2 score)							
	C		$D_{net,ori}$		ΔD_{net}		V_{bump}	
	Train	Test	Train	Test	Train	Test	Train	Test
PlanA	0.8963	0.8529	0.8652	0.8475	0.8765	0.8584	0.8356	0.8265
PlanB	0.9265	0.8958	0.9325	0.9014	0.8554	0.8397	0.8256	0.8446
PlanC	0.9768	0.9685	0.9587	0.9886	0.9875	0.9648	0.9256	0.9584
PlanD	**0.9997**	0.9957	**0.9968**	0.9981	0.9957	0.9967	0.9611	0.9684
PlanE	0.9985	**0.9958**	0.9958	**0.9986**	**0.9968**	**0.9985**	**0.9865**	**0.9784**

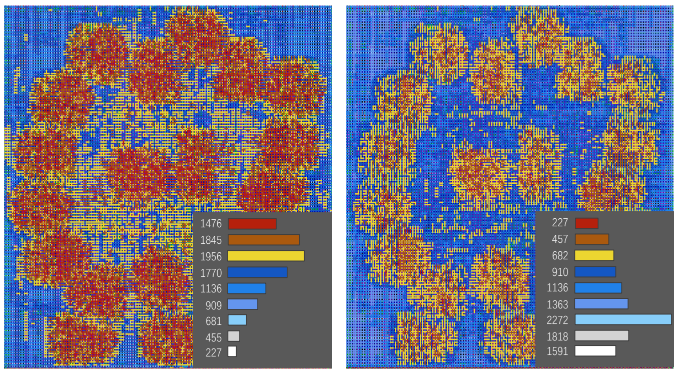
by XGBoost, GNNTimer, and GraphCAD are adopted, and the same prediction methodology as our model is applied, excluding timing-window-related features. The results indicate that both XGBoost and GNNTimer exhibit relatively low accuracy, which appears to be heavily impacted by the absence of timing-window-related features, which are crucial for capturing crosstalk effects that occur only when both spatial proximity and timing overlap conditions are met. In contrast, our proposed model leverages these features to achieve predictions closely aligned with post-routing timing results.

To validate the path-level accuracy, TABLE V presents the prediction errors of WNS and TNS. The results indicate that both XGBoost and GNNTimer exhibit limited accuracy when RC parameters are unavailable. GNNTimer, specializing in end-to-end predictions, achieves higher accuracy in predicting WNS and TNS compared to XGBoost. Conversely, XGBoost is more suitable for predicting individual net-level crosstalk delay, thus slightly outperforming GNNTimer in the net-level delay predictions shown in TABLE IV. GraphCAD significantly improves prediction accuracy by leveraging accurate parasitic parameter information. Nevertheless, since such parasitic information is only available after full routing and extraction, GraphCAD cannot provide early-stage guidance during the routing process. In contrast, our proposed model achieves delay predictions closely aligned with post-routing timing results without relying on parasitic parameter files, highlighting its practical advantage in early-stage timing prediction.

D. Efficiency and Scalability Analysis

To evaluate the efficiency of our proposed approach, TABLE V provides a comparative analysis of runtime among the baseline (which includes complete routing, parasitic extraction, and STA), XGBoost, GNNTimer, GraphCAD, and ours. All ML-based approaches demonstrate significantly reduced runtimes compared to the baseline. Specifically, XGBoost and GNNTimer achieve rapid execution speeds due to their reliance solely on placement results, thereby eliminating the need for routing and substantially reducing data preparation time.

In contrast, GraphCAD requires the completion of parasitic parameter extraction prior to prediction, resulting in longer data preparation times and thus less favorable runtime performance. Our proposed model, which necessitates only an initial global routing solution, effectively balances agility and accuracy. By leveraging timing-window-related features and the physical



(a) Without our model (b) With our model

Fig. 6: Comparison of the crosstalk-induced delay distribution with and without the embedding of our model.

information of nets, our model achieves crosstalk delay predictions that closely match signoff-level accuracy, yielding results comparable to those of GraphCAD. This advantage becomes particularly pronounced in large-scale circuit scenarios, as shown in TABLE V, where XGBoost and GNNTimer exhibit notably increased prediction errors, and GraphCAD experiences significantly prolonged data preparation times. Consequently, our approach demonstrates clear superiority in terms of both runtime efficiency and prediction accuracy, especially when applied to large-scale circuits.

VI. Case Study on AES_Core Benchmark

To validate the efficiency of our crosstalk-aware timing prediction model, we conducted comparative analyses using two distinct schemes applied to the same design (`aes_core`). The evaluation workflow comprises four sequential phases: (1) global routing; (2) GNN-model-guided crosstalk net identification; (3) crosstalk net optimization; and (4) detailed routing, parasitic extraction, and signoff timing analysis. The experimental pipeline (phases 1–3) initiates with an initial global routing solution generated by FastRoute [18]. Following crosstalk optimization, phase 4 completes the implementation flow using commercial EDA tools: detailed routing via ICC2, parasitic extraction via StarRC, and sign-off timing verification via PrimeTime-SI. For consistency, phase 4 employs a fixed 10-round iteration, which was empirically chosen to balance convergence and runtime. Then proceed with the detailed description of Scheme 1 and Scheme 2.

- **Scheme 1 (Fixed Utilization Crosstalk Optimization)**: Maintains constant routing density at 70% to prioritize crosstalk mitigation.
- **Scheme 2 (Density-Scalable Timing-Driven Implementation):** Begins with 50% routing density and iteratively compresses layout until reaching either Design Rule Check (DRC) violations or timing margin exhaustion.

As shown in TABLE VI, both Scheme 1 and Scheme 2 are implemented using our full flow, which integrates the proposed crosstalk-aware timing prediction model into the global routing. Compared to the baseline FastRoute-only implementation, **Scheme 1** achieve improvements: 80.9% average reduction in

TABLE IV: Prediction accuracy (R^2 score) of different models compared to PrimeTime-SI results.

Benchmarks	Wire capacitance C (%)				No-crosstalk delay $D_{net,ori}$ (%)				Crosstalk-induced delay ΔD_{net} (%)				Crosstalk-induced worst bump V_{bump} (%)			
	XGBoost [3]	GNNTimer [9]	GraphCAD [10]	**Ours**	XGBoost [3]	GNNTimer [9]	GraphCAD [10]	**Ours**	XGBoost [3]	GNNTimer [9]	GraphCAD [10]	**Ours**	XGBoost [3]	GNNTimer [9]	GraphCAD [10]	**Ours**
pci	83.42	78.65	99.12	**98.87**	81.76	84.93	99.45	**98.92**	75.34	58.12	**98.38**	98.36	85.67	86.56	**99.67**	98.26
spi	81.27	76.88	98.84	**98.45**	79.63	81.12	99.13	**98.74**	72.91	54.67	**98.92**	98.89	82.98	78.67	**99.45**	99.01
systemcdes	80.94	70.45	**98.67**	98.13	82.34	69.78	**98.89**	98.42	73.56	49.23	**98.45**	98.41	86.67	85.67	98.45	**99.02**
mem_ctrl	85.63	79.34	**98.23**	97.87	80.12	75.45	98.56	**98.34**	76.89	59.12	**97.78**	97.76	92.56	69.56	99.02	**99.21**
usb_funct	79.45	75.89	**97.34**	96.87	76.78	73.56	**97.45**	97.12	69.87	51.34	**96.92**	96.91	88.56	78.98	97.46	**98.34**
ethernet	77.12	71.23	**96.45**	96.12	73.45	69.87	**96.67**	96.45	67.34	43.56	**96.45**	96.44	89.45	72.97	97.24	**97.27**
wb_conmax	75.89	69.78	**95.78**	95.45	71.23	67.89	**95.34**	95.12	65.12	39.87	**95.56**	95.55	90.45	85.31	**98.11**	97.67
Average	80.53	74.60	97.92	**97.54**	77.47	74.37	97.93	**97.59**	71.15	50.27	**97.78**	97.77	88.05	79.67	**98.49**	98.40

TABLE V: Comparison on time prediction performance (WNS and TNS) and runtime. **Note:** The baseline runtime includes routing, RC extraction, and STA; in contrast, XGBoost and GNNTimer only perform the prediction step, GraphCAD conducts routing and RC extraction prior to prediction, and our work conducts global routing and the prediction step.

Benchmarks	Errors of WNS (%)				Errors of TNS (%)				**Runtime** (s)				
	XGBoost [3]	GNNTimer [9]	GraphCAD [10]	**Ours**	XGBoost [3]	GNNTimer [9]	GraphCAD [10]	**Ours**	Baseline [8]	XGBoost [3]	GNNTimer [9]	GraphCAD [10]	**Ours**
pci	20.54	19.36	5.21	**3.08**	18.89	20.42	**4.85**	3.43	6.05	1.42	0.52	5.02	**1.94**
spi	23.69	21.35	**4.03**	4.53	21.79	18.56	**4.13**	4.41	7.78	1.82	0.58	4.36	**2.30**
systemcdes	16.41	18.52	5.35	**4.82**	18.58	19.63	5.20	**4.37**	25.56	1.98	0.66	19.36	**2.56**
mem_ctrl	18.74	19.64	4.16	**3.31**	16.85	17.36	4.98	**3.85**	32.99	1.81	0.97	21.35	**3.61**
usb_funct	17.17	16.58	**3.18**	3.55	15.49	17.25	**2.86**	4.93	59.85	1.95	1.28	50.23	**6.72**
ethernet	22.63	19.35	**3.42**	3.76	17.71	16.85	**3.05**	4.12	189.23	1.34	1.49	172.45	**15.07**
wb_conmax	27.70	22.35	3.29	**3.21**	20.09	16.37	**2.57**	2.96	219.63	2.56	2.03	201.35	**22.36**
Average	20.98	19.59	4.23	**3.75**	18.49	18.06	**3.95**	4.01	77.29	1.84	1.07	67.73	**7.79**
Speedup ($\times$)	–	–	–	–	–	–	–	–	1.00	42.02$\times$	72.24$\times$	1.14$\times$	**9.92**$\times$

TABLE VI: Post-implementation results of aes_core with and without crosstalk-aware optimization.

Metric	**FastRoute Only**	**FastRoute + our Model (Full Flow)**	
	No Optimization	**Scheme 1**	**Scheme 2**
Chip area (μm^2)	4968	4968	4322
Cell area (μm^2)	4195	4195	3478 (17%)
Frequency (MHz)	468	562 (28%)	557 (19%)
WNS (ns)	-0.83	-0.65 (21%)	-0.64 (23%)
TNS (ns)	-192.3	-146.9 (24%)	-186.6 (3%)
Total delta delay (ns)	4.93	0.94 (81%)	3.57 (27%)
Wirelength (m)	0.318	0.314	0.435
Total power (mW)	4.68	4.63	4.92
Total runtime (s)	42.68	8.60	40.63

crosstalk delay, 21.7% improvement in WNS, and 1.2 % power benefit ($P_{\text{total}} = P_{\text{dyn}} + P_{\text{stat}}$). Crosstalk mitigation inherently necessitates wire routing adjustments, resulting in a slight increase in the overall design wire length as an optimization cost. Fig. 6 illustrates the crosstalk delay distribution before and after optimization, with an inset detailing the statistical count of nets exhibiting significant crosstalk delay. Specifically, red indicates nets with high *delta delay*, while blue and white represent nets with successively lower crosstalk delay. The blue and white regions denote nets with relatively minor crosstalk delay. Our crosstalk-aware timing prediction model identifies crosstalk-sensitive nets and guides the router to optimize those nets. The number of nets severely affected by crosstalk decreased from 1476 to 227.

In **Scheme 2**, with a final core utilization 85%, aes_core can realize significant 17% area reduction and 19% improvement in frequency, as shown in TABLE VI. The increase in wirelength and power consumption is negligible in comparison to the benefit to the area. This case study confirms that our model effectively guides routing engines to prioritize signal integrity constraints without compromising routability or runtime efficiency.

VII. Conclusion

In this paper, we presented ChronoTE, an innovative edge-enhanced graph neural network for crosstalk-aware timing estimation. By integrating physical topology with timing-window-related features, ChronoTE achieves signoff-quality predictions in global routing without requiring full routing or parasitic extraction. Experiments demonstrate reduced post-routing corrections and enhanced PPA metrics. Future work will explore further optimizations and dynamic parameter integration to enhance scalability and prediction fidelity for advanced VLSI design closure.

VIII. Acknowledgments

The research work described in this paper was conducted in the JC STEM Lab of Intelligent Design Automation funded by The Hong Kong Jockey Club Charities Trust. This work is jointly supported by the Research Grants Council of Hong Kong SAR (No. CUHK14211324).

References

[1] M. Brunion, A. Sharma, G. Mirabelli, D. Abdi, Y. Zhou, H. Kükner, O. Zografos, F. G. Redondo, D. Biswas, G. Hellings, J. Ryckaert, and J. Myers, "System technology co-optimization of cost-bandwidth tradeoffs in network on chip through 3D integration and backside signals," in *IEEE International Electron Devices Meeting (IEDM)*, pp. 1–4, 2024.

[2] 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)*, pp. 63–69, 2022.

[3] R. Liang, Z. Xie, J. Jung, and et al., "Routing-free crosstalk prediction," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–9, 2020.

[4] M. S. H. Omshi, R. F. Mirzaee, A. Reza, and M. Mirzaei, "Low-power bus encoding by ternary LWC and quaternary transition signaling: From initial concept to circuit design," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 32, no. 4, pp. 682–694, 2023.

[5] V. Huang, D. Shim, H. Simka, and A. Naeemi, "From interconnect materials and processes to chip level performance: Modeling and design for conventional and exploratory concepts," in *IEEE International Electron Devices Meeting (IEDM)*, pp. 32.6.1–32.6.4, 2020.

[6] D. Wu, J. Hu, R. Mahapatra, and M. Zhao, "Layer assignment for crosstalk risk minimization," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pp. 159–162, 2004.

[7] J. Liu, C.-W. Pui, F. Wang, and E. F. Y. Young, "CUGR: Detailed-routability-driven 3D global routing with probabilistic resource model," in *ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.

[8] "PrimeTime SI: Crosstalk delay and noise.," 2022. https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html/.

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

[10] F. Liu, G. Guo, Y. Ye, Z. Wang, W. Fu, W. Sheng, and B. Yu, "GraphCAD: Leveraging graph neural networks for accuracy prediction handling crosstalk-affected delays," in *ACM International Symposium on Physical Design (ISPD)*, p. 125–133, 2025.

[11] 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)*, pp. 1–8, 2015.

[12] V. A. Chhabria, W. Jiang, A. B. Kahng, and S. S. Sapatnekar, "From global route to detailed route: ML for fast and accurate wire parasitics and timing prediction," in *ACM/IEEE Workshop on Machine Learning CAD (MLCAD)*, pp. 7–14, 2022.

[13] H. Ren, S. Nath, Y. Zhang, H. Chen, and M. Liu, "Why are graph neural networks effective for eda problems?," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2022.

[14] A. Ghose, V. Zhang, Y. Zhang, D. Li, W. Liu, and M. Coates, "Generalizable cross-graph embedding for gnn-based congestion prediction," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–9, 2021.

[15] B. Onal, E. Dogan, M. H. Khan, and M. R. Guthaus, "GAT-Steiner: Rectilinear steiner minimal tree prediction using GNNs," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024.

[16] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[17] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," in *International Conference on Learning Representations (ICLR)*, 2020.

[18] Y. Xu, Y. Zhang, and C. Chu, "Fastroute 4.0: Global router with efficient via minimization," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pp. 576–581, 2009.

[19] T. Ajayi, V. A. Chhabria, M. Fogaça, and et al., "Toward an open-source digital flow: First learnings from the OpenROAD project," in *ACM/IEEE Design Automation Conference (DAC)*, 2019.

[20] "OpenCores.," 2021. https://opencores.org/.

[21] "Synopsys design compiler user guide," 2022. Guide,http://www.synopsys.com.

[22] "Synopsys IC. compiler II user guide.," 2022. Guide,http://www.synopsys.com.