PRO-TIME: Prerouting Optimization-Aware Timing Prediction via Multimodal Learning

Ziyi Wang, Siting Liu, Yuan Pu, Song Chen, *Member, IEEE*, Tsung-Yi Ho, *Fellow, IEEE*, and Bei Yu, *Senior Member, IEEE*

***Abstract*—Fast and accurate prerouting timing prediction is crucial in the very-large-scale integration (VLSI) design flow. Existing machine learning (ML)-assisted prerouting timing evaluators neglect the impact of timing optimization, which may render their approaches impractical in real circuit design flows. To address the challenges posed by timing optimization, we propose PRO-TIME, a prerouting optimization-aware timing prediction framework that is driven by multimodal learning. Specifically, we propose a novel endpoint embedding framework that integrates both netlist and layout information. A customized graph neural network (GNN) model is used for extracting endpoint-wise netlist information, which is motivated by the delay propagation process. Meanwhile, we apply the U-net model with a masking strategy to extract endpoint-wise layout information. Furthermore, we propose an adaptive layout mask adjustment scheme to boost performance by leveraging the layout information more effectively. Comprehensive experiments on large-scale RISC-V designs with advanced 7-nm technology node demonstrate the superiority of our model compared to the state-of-the-art prerouting timing evaluators.**

***Index Terms*—Convolutional neural networks, deep learning, design automation, graph neural networks, physical design, timing.**

I. Introduction

The BREAKTHROUGH in advanced technology nodes has sparked interest in the circuit design flow, as it dramatically increases the circuit size and brings new challenges to meet the design requirements, e.g., timing constraints. As the two most important processes, placement allocates positions to the gates, while routing is used to obtain the precise interconnection between pins. To meet the sign-off timing constraints, the processes of placement and routing (PnR) often necessitate multiple iterations, which can be exceedingly time-consuming. Consequently, prerouting timing prediction based on the placement solution has drawn researchers' attention to save the abundant routing runtime and provide quick feedback to optimize timing early.

Received 30 September 2024; revised 25 February 2025; accepted 6 May 2025. Date of publication 12 May 2025; date of current version 24 November 2025. This work was supported in part by The Research Grants Council of Hong Kong SAR under Grant CUHK14211824, Grant CUHK14201624, and Grant CUHK14210723; and in part by the MIND Project under Grant MINDXZ202404. This article was recommended by Associate Editor L. Behjat. *(Corresponding author: Bei Yu.)*

Ziyi Wang, Siting Liu, Yuan Pu, Tsung-Yi Ho, and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR (e-mail: byu@cse.cuhk.edu.hk).

Song Chen is with the School of Microelectronics, University of Science and Technology of China, Hefei 230026, China.

Digital Object Identifier 10.1109/TCAD.2025.3569488

Efficient and accurate prerouting timing estimation is necessary for timing-driven placement engines [1]. In practice, the linear *RC* static timing analysis (STA) model, Elmore's model [2], is frequently employed to facilitate swift timing evaluations using only placement results. However, the precision of Elmore's model is often questioned due to its reliance on inaccurately estimated wirelength, which lacks the detailed insights that actual routing information provides [3].

The integration of trending machine learning (ML) techniques has unveiled new avenues for enhancing prerouting timing evaluation, as evidenced by recent research [4], [5], [6]. Generally, these approaches adopt a local-view methodology that relies on the prediction of local net/cell delays, which forms the foundation for further prediction of global timing performance, e.g., endpoint arrival time. These can be broadly categorized into two distinct types: 1) two-stage approach and 2) end-to-end approach. The two-stage approaches [4], [5] first predict local net/cell delays and then apply PERT traversals [7] to assess global timing metrics (i.e., endpoint arrival time). On the other hand, the end-to-end method [6] can directly predict the global timing performance in a single run. This method leverages the generation and propagation of net/cell embeddings following a topological order, culminating in predictions at the endpoints. Nevertheless, it still relies on local net/cell delay prediction as auxiliary tasks. Overall, these studies not only underscore the critical need for effective prerouting timing prediction but also highlight the successful application of ML-assisted methods in addressing this challenge.

Despite the advances brought by ML techniques in prerouting timing evaluation, a critical aspect often overlooked in previous studies is the consideration of timing optimization. Specifically, they set their prediction target as the post-routing timing performance of a design flow that skips the timing optimization stage. Although this simplification significantly reduces the prediction difficulty, it makes these works impractical as timing optimization is a necessary step in modern design flows and has a non-negligible impact on sign-off timing performance, as shown in Table I.

However, optimization-aware prerouting timing prediction is nontrivial, and what we need to consider is not just a simple shift of the ground-truth labels. The real key challenge brought by timing optimization is a phenomenon that we term "feature mismatching," which is caused by netlist restructuring that happens during timing optimization. Fig. 1 gives an example of gate rewriting, which is a frequently used timing optimization technique. Here, the multiple-input

1937-4151 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.

TABLE I

STATISTICS OF THE DATASET, WHERE THE LEFT COLUMNS ILLUSTRATE THE INPUT DESIGN (BEFORE ROUTING), AND THE RIGHT COLUMNS ILLUSTRATE THE IMPACT OF TIMING OPTIMIZATION MEASURED BY THE CHANGE RATIO BETWEEN SIGN-OFF TIMING METRICS GENERATED BY FLOWS WITH/WITHOUT TIMING OPTIMIZATION. #REPLACED DENOTES THE PERCENTAGE OF INPUT NET/CELL EDGES REPLACED DURING TIMING OPTIMIZATION. ΔDELAY IS CALCULATED UPON THE UNREPLACED NETS/CELLS. FURTHERMORE, IN THE LEFT COLUMNS, EDP STANDS FOR ENDPOINT, e_n AND e_c DENOTE NET EDGE AND CELL EDGE, REPRESENTING THE CONNECTION BETWEEN NET DRIVE/SINK PINS AND CELL INPUT/OUTPUT PINS, RESPECTIVELY

Benchmark		Input information				Impact of timing optimization on sign-off metrics					
						slack reduction		net variation		cell variation	
		#pin	#edp	$\#e_n$	$\#e_c$	Δwns	Δtns	#replaced	Δdelay	#replaced	Δdelay
train	jpeg	932842	40801	650878	607795	98.8%	99.8%	32.5%	50.8%	35.4%	40.6%
	rocket	698347	52731	490499	432068	94.0%	94.8%	28.5%	70.0%	8.0%	24.2%
	smallboom	694441	61764	488052	423344	87.4%	99.1%	40.9%	53.1%	15.6%	39.8%
	steelcore	26598	1662	19439	17732	89.6%	98.3%	49.8%	51.8%	18.4%	29.4%
	xgate	20842	684	14653	13010	94.8%	99.1%	31.3%	50.7%	16.9%	20.9%
test	arm9	44469	2500	33065	29287	82.5%	88.8%	46.7%	47.8%	24.0%	42.9%
	chacha	35687	1986	25117	23083	86.5%	88.0%	47.1%	62.5%	38.8%	40.2%
	hwacha	1357798	61313	985057	922085	91.7%	92.5%	45.1%	70.4%	22.0%	37.1%
	or1200	1165114	172401	844443	658961	95.4%	97.3%	49.1%	61.4%	20.8%	28.6%
	sha3	794720	60323	552021	485596	96.0%	97.4%	30.3%	77.6%	8.3%	28.8%
Avg	train	474614	31528	332704	298790	92.9%	98.2%	36.6%	55.3%	18.9%	31.0%
	test	679558	59705	487941	423802	90.4%	92.8%	43.7%	63.9%	22.8%	35.5%

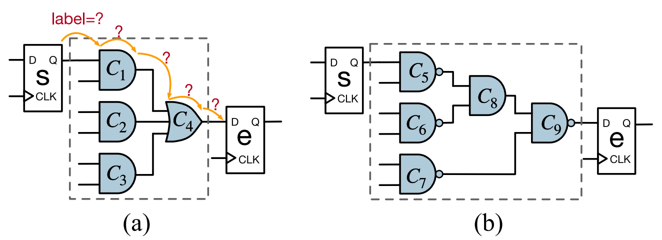

Fig. 1. Example of feature mismatching caused by timing optimization, where the subnetlist within the dotted box is replaced to improve timing. This leads to a mismatch between input features used for prediction, e.g., information about C_1–C_4, and ground-truth features, e.g., information about C_5–C_9. (a) Input netlist. (b) Optimized netlist.

gates in the dotted box are replaced by more efficient gates for better timing performance. This modification in the netlist structure poses significant challenges to timing prediction. Specifically, the input features utilized for delay prediction in the restructured subnetlists—for instance, information from C_1 to C_4—no longer align with the actual, or ground-truth features, i.e., information from C_5 to C_9.

The discrepancy in features leads to inaccuracies in local delay predictions within the restructured regions. Particularly, the more accurately the models are able to predict delays in unrestructured regions, the less accurately they perform in restructured regions. In other words, local delay supervision is no longer reliable due to feature mismatching. As the error accumulates along the timing paths, this eventually leads to inaccurate prediction of endpoint arrival time (global timing metrics). It is found that feature mismatching happens frequently, as on average 40% of nets and 21% of cells are replaced during timing optimization (details in Table I). Such misalignment prohibits the local-view fashion adopted by previous works [4], [5], [6] from fitting the real-world scenario where timing optimization is taken into account. Consequently, it arouses the demand for developing new methodologies to handle the impact of timing optimization.

Based on the fact that timing endpoints are never replaced during timing optimization and the previous finding that it is easier to model timing optimization's impact globally than locally [8], we propose to take a global endpoint-wise view to build the framework. Furthermore, since most timing optimization techniques include gate insertion or gate sizing, placement should reserve space for subsequent timing optimization. In other words, the timing optimizer's efficacy is tied closely to global layout information [9]. Therefore, layout information benefits modeling the impact of timing optimization. Based on the above observations, we develop a novel endpoint-wise prerouting timing prediction flow via multimodal fusion. A customized graph neural network (GNN) model is utilized to extract endpoint-wise netlist information, which reflects the basic timing condition. Meanwhile, we apply a U-net model and a novel layout masking technique to extract endpoint-specific layout information, which reflects the impact of timing optimization. The extracted information from the two modalities is then fused to make the final prediction on global timing performance, i.e., endpoint arrival time. Furthermore, we propose an adaptive mask adjustment scheme to better reflect the individual impact of timing optimization on each endpoint.

The major contributions of this article are listed as follows.

1) To the best of our knowledge, we are the first to provide multimodal fusion-based optimization-aware prerouting timing prediction flow.
2) A customized GNN is presented to extract and aggregate endpoint-wise netlist information.
3) A U-net model with an endpoint-wise masking technique is developed to efficiently extract the unique layout information for each timing endpoint.
4) We propose a scheme that alternatively adjusts the endpoint masks to better reflect the individual impact of layout information on each timing endpoint.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.

5) We conduct experiments on large-scale RISC-V designs in the advanced 7-nm node to confirm the effectiveness of our proposed techniques.

The remainder of this article is organized as follows. Section II introduces the problem definition and the background of timing optimization techniques and learning models. Section III overviews our proposed end-to-end endpoint embedding driven timing prediction flow. Section IV-B describes extracting endpoint-wise netlist information. Section V introduces the proposed endpoint-wise layout information extraction technique. Section VI introduces our adaptive layout mask adjustment scheme. Section VII presents experimental results, followed by the conclusion in Section VIII.

II. Preliminaries and Problem Definition

A. Timing Optimization

Timing optimization is a critical phase in the electronic design automation (EDA) process that focuses on improving the timing performance of a digital circuit. As technology nodes shrink and circuits become more complex, ensuring that the circuit meets its timing requirements becomes increasingly challenging.

As listed in Table I, timing optimization significantly impacts both local and global timing performance. It is worth noting that while the overall goal of optimization is to improve the global timing performance, the effects on individual nets or cells can be unpredictable. For instance, the delay on certain nets may increase compared to those produced by a flow without optimization, whereas others may be significantly reduced. This uncertainty stems from the complex global interactions among nets, cells, and limited layout resources [8], making it exceedingly challenging to accurately predict local net or cell delays when considering the impact of timing optimization.

Timing optimization techniques can be categorized into two types: 1) structure-preserved and 2) structure-destructed. The structure-preserved techniques [10] enhance timing without altering the netlist structure. A representative technique is gate-sizing [11], which involves optimizing transistor sizes within each gate to improve its delay or drive strength. On the other hand, structure-destructed techniques [12], e.g., logic restructuring, achieve timing optimization by modifying the netlist structure. Such modifications, while preserving the circuit's functionality, may involve adding/deleting gates or reconfiguring connections between existing gates to enhance driving strength and signal integrity [12].

B. Graph Neural Network

GNNs have gained prominence as a powerful method for processing graph-structured data and extracting valuable insights from graph information in recent years [13], [14], [15]. They operate based on an iterative message-passing scheme that captures the structural relationships within the neighborhoods of nodes. Let $G = \langle \mathcal{V}, \mathcal{E} \rangle$ denote a graph, where $\mathcal{V} = \{v_1, v_2, , \ldots, v_n\}$ denote the set of nodes, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ represent the set of edges connecting the nodes. In a GNN with k layers, the propagation at the kth layer is described by

$$\begin{aligned} \vec{a}_v^{(k)} &= \text{AGGREGATE}\Big(\{\vec{h}_u^{(k-1)} : u \in \mathcal{N}(v)\}\Big) \\ \vec{h}_v^{(k)} &= \text{COMBINE}\Big(\vec{a}_v^{(k)}, \vec{h}_v^{(k-1)}\Big). \end{aligned} \tag{1}$$

Here, $\vec{a}_v^{(k)}$ and $\vec{h}_v^{(k)}$ denote the aggregated message from neighboring nodes and the embedding of vertex v at the kth layer, respectively, and $\mathcal{N}(v)$ represents the set of neighboring nodes of v. The AGGREGATE function is responsible for collecting messages from a node's neighborhood, and the COMBINE function integrates the node's previous representation with the incoming messages.

The exploration of GNNs offers numerous opportunities to address challenges in EDA by leveraging graph representations [16], [17]. This can be attributed to the graph nature of much raw data in the EDA flow, e.g., analog circuit, netlist, RTL, etc. For instance, Google [18] proposes a floorplanning framework with an edge-based GNN. The work of [19] applies a multistage GNN for the testing points-insertion flow. Besides, Wang et al. [20] incorporated GNN with contrastive learning for netlist functionality extraction.

C. Convolutional Neural Network

The convolutional neural network (CNN) is one of the most significant deep learning models for retrieving local–global information of Euclidean data (grid-pattern data) [21], [22], [23]. CNNs exploit spatial correlations in data by applying convolutional filters that perform convolution operations as they slide over input features to produce feature maps. The architecture typically consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers (FCNs). The fast inference and excellent information fusion capabilities of CNN models have fueled the growth of modern EDA tools in recent years. The applications of CNNs span the whole very-large-scale integration (VLSI) design flow, from the logic synthesis stage [24] to the whole physical design flow [25], [26], [27], [28], [29]. In addition, many successful attempts have been made in the fabrication stage [30], [31], [32], [33]. Researchers are continually searching for new opportunities to exploit the power of CNNs to improve the overall performance of EDA tools.

D. Multimodal Learning

In the rapidly evolving landscape of ML, multimodal learning has emerged as a pivotal paradigm [34], [35]. Traditional ML models typically handle a single data type, such as text or images. However, human interaction with the environment involves simultaneous processing of multiple senses, which provides a more comprehensive understanding of the surrounding context. Multimodal learning algorithms strive to replicate this integrative approach by combining information from different sources (e.g., text, video, images, and graphs). This method reflects the inherent complexity of the real world, where data is not limited to one modality. In recent years, the abundance of easily accessible data from different aspects has offered the prospect of multimodal fusion in EDA.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.

E. Problem Definition

In this part, we first introduce some related terms and then give the problem definition.

Definition 1 (Netlist): The gate-level netlist of an electric circuit includes a list of circuit components, e.g., AND gates, and describes the nets among them. It can be represented as a graph.

Definition 2 (Layout): The chip layout is obtained by assigning each circuit component a specific location. Layout information can be formulated into a grid-pattern region with a set of evenly distributed horizontal and vertical lines.

Definition 3 (Timing Path): A timing path comprises a startpoint, an endpoint, and the combinational logic network in the middle. The startpoint should be either a primary input (PI) or the clock pin of a sequential element, and similarly, the endpoint should be either a primary output or the input pin of a sequential element. The slack of a timing path is defined as the difference between the required time for the signal to arrive at the endpoint and the actual time it takes, which represents the criticality of the timing path/endpoint.

Problem 1 (Optimization-Aware Prerouting Timing Prediction): Given the prerouting layout and netlist of a design, our goal is to make an accurate and efficient estimation of the post-routing global timing metrics, i.e., endpoint arrival time, with the impact of timing optimization taken into account.

III. Flow Overview

As mentioned before, the impact of timing optimization, particularly through netlist restructuring, poses significant challenges to accurate prerouting timing prediction. Based on the observation that many commonly used timing optimization techniques rely on global insights and layout information, we present an endpoint embedding flow that models the overall timing performance by combining information from the following two aspects.

1) *Netlist:* The information from netlists is necessary since delay computation is based on the connections between nets/cells. With netlist information, we can model the essential timing condition.
2) *Layout:* The layout information plays a dominant role in determining the timing optimizer's impact since most optimization techniques need space to be applied [9].

The overview of our end-to-end framework, namely, PRO-TIME, is shown in Fig. 2, which follows a multimodal fusion paradigm, i.e., combining information from multiple modalities. For each timing endpoint e, we encode its netlist information into the netlist embedding $\vec{h}_e^n$ with the help of a customized GNN model. Meanwhile, a U-net model and a novel masking technique are utilized to embed global layout information into the endpoint-wise layout embedding $\vec{h}_e^l$. The final embedding $\vec{h}_e$ for the endpoint e is given by a combination operation (e.g., concatenation) of $\vec{h}_e^n$ and $\vec{h}_e^l$. A multilayer perceptron (MLP) regression model is then applied to consume the generated endpoint embeddings for predicting global timing metrics, which can be written as follows:

$$\hat{a}_e = f^r\left(\vec{h}_e^n, \vec{h}_e^l\right) \tag{2}$$

where $\hat{a}_e$ denotes the predicted arrival time of endpoint e, f^r represents the modality-fusion model used to fuse the netlist and layout embeddings and output the prediction, which is instantiated as a vector concatenation operator followed by an MLP-based regression model. We utilize the mean-squared error (MSE) as the loss function to guide training

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}\left(\vec{a}_i - \hat{a}_i\right)^2 \tag{3}$$

where $\vec{a}_i$ and $\hat{a}_i$ are the ground-truth arrival time and predicted arrival time for the ith endpoint, respectively. The algorithm details are introduced in the following sections.

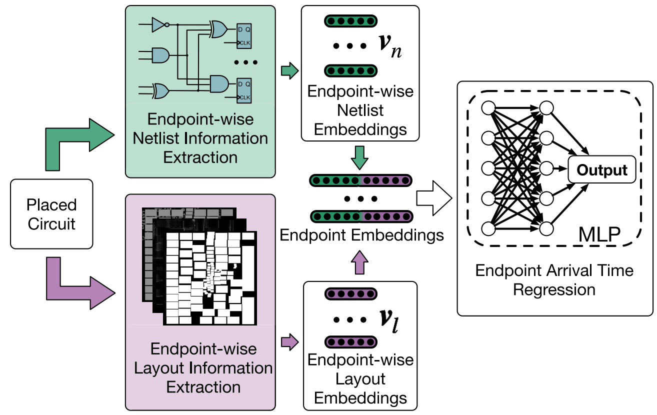

Fig. 2. Overview of our proposed end-to-end endpoint embedding framework PRO-TIME, which generates endpoint-wise embeddings and conducts prediction based on the embeddings. We apply a customized GNN model to extract information from netlists and a U-net model with a novel masking technique to extract information from layouts. The extracted information is then fused to generate the final embedding for each timing endpoint.

IV. Netlist Information Extraction

We first discuss how to extract timing information from netlists and embed it into a single netlist embedding for each timing endpoint. Based on the graph nature of netlists, we apply the GNN as the netlist encoder, which has demonstrated great power in processing graph-structured data.

A. Heterogeneous Graph Construction

Given a netlist, we first preprocess it by transforming it into a proper graph format, which then serves as the input to our customized GNN model. Based on the fact that the pin is the essential element in timing analysis, each pin within the netlist is treated as a node, and the connections between these pins are represented as edges. Furthermore, considering that there are two types of timing arcs, we model the netlist as a heterogeneous graph with two edge types: 1) the net edge and 2) the cell edge. Net edges represent the connection between a net's drive pin and its respective sink pins. On the other hand, cell edges connect a cell's input pins to its output pin.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.

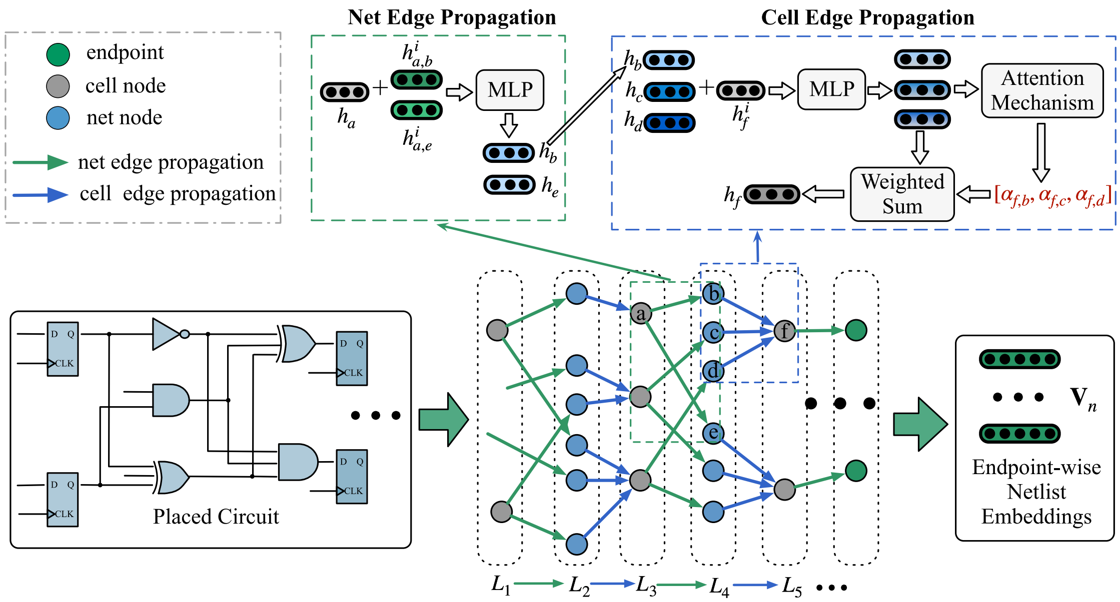

Fig. 3. Message-passing scheme of our customized GNN, where the black dotted boxes show the graph's topological levels. L_i represents the ith topological level, h_v represents the embedding of node v, h_v^i denotes the initial cell features of node v, $h_{u,v}^i$ represents the initial net features along the edge (u, v), and $\alpha_{u,v}$ gives the attention coefficient. The information flows in the topological order, aggregated at the endpoints to generate endpoint-wise netlist embeddings.

Hereafter, we refer to the output of cell edges as cell nodes and the sink of net edges as net nodes. All edges are directed, and by removing the cell edges between the data input pin and data output pin of sequential elements (e.g., register), there are no cycles in the graph. Consequently, the constructed graph can be seen as a directed acyclic graph (DAG). Fig. 3 gives an example of the transformed graph for a given circuit.

As for features, we select the most relevant ones for timing.

1) *Net Distance:* The Manhattan distance between the positions of a net's drive pin and its sink pin, playing a dominant role in the net delay.
2) *Cell Driving Strength:* Extracted by the cell type name, determining the output impedance of the driver.
3) *Gate Type:* For example, AND and XOR are embedded as a one-hot vector.
4) *Pin Capacitance:* Extracted from the timing library. We treat the net distance attribute as the edge feature for net edges and attach the other three cell features to the cell nodes.

B. Customized Graph Neural Network

GNN techniques [16] have emerged as a powerful paradigm for extracting insights from graph-structured data. Given that netlists can naturally be represented as graphs, the application of GNNs to generate endpoint-wise netlist embeddings appears to be a promising approach [36], [37], [38]. Despite this, it is crucial to tailor these models to properly capture the timing condition. Motivated by the process of delay propagation, our customized GNN model is designed to propagate information in the topological order, culminating at the timing endpoints.

Fig. 3 illustrates the process of generating netlist embeddings for each timing endpoint using our specialized GNN model. The message-passing process commences at the PIs and propagates through the graph level by level until all endpoints have been reached. To discriminate between the two timing arcs, we design two different message propagation schemes for cell and net edges, respectively.

For cell edges, we consider a multiple-to-one relationship since each cell output pin might be driven by multiple input pins. The embedding of a cell node v is composed of two parts: the aggregated messages (or embeddings) from its predecessor net nodes and the intrinsic attributes of the cell itself (referred to as cell features). Based on the fact that the arrival time at the output pin is predominantly influenced by the maximum arrival time among the input pins, we incorporate an attention mechanism [39], [40] to selectively emphasize the most critical predecessor. For a given cell node v, we calculate the attention coefficient for each of its predecessor nodes u as (4). This coefficient reflects the significance of the embedding of node u in the final embedding of node v

$$e_{v,u} = \vec{a}^{\top}\big(\mathrm{MLP}^a(h_v^i, h_u)\big). \tag{4}$$

Here, $\vec{a}$ is a shared learnable vector, $\cdot^{\top}$ represents transposition, MLP^a is a multiplayer perceptron, h_v^i denotes the initial cell feature of node v, and h_u denotes the generated embedding of node u. To ensure that these attention coefficients are comparable across different nodes, we normalize them using the softmax function, as detailed in

$$\alpha_{v,u} = \mathrm{softmax}_v\big(e_{v,u}\big) = \frac{\exp\big(e_{v,u}\big)}{\sum_{k\in\mathcal{N}(v)}\exp\big(e_{v,k}\big)}. \tag{5}$$

Here, $\mathcal{N}(v)$ denotes the set of predecessor nodes of v. The resulting normalized attention coefficients are then employed to compute a weighted linear combination of the predecessor

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.

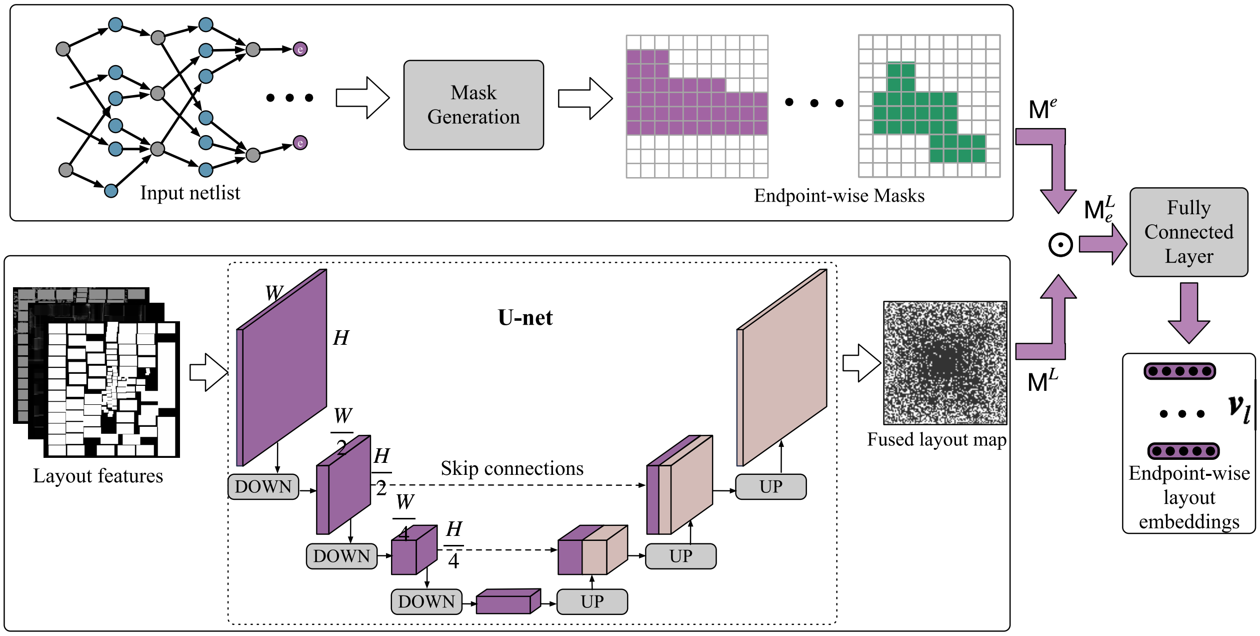

Fig. 4. Our endpoint-wise layout embedding generation flow with a U-net model and a novel endpoint-wise masking technique. Our proposed masking technique generates a unique layout embedding for each timing endpoint by capturing the endpoint's individual layout impact.

messages, which forms the final output embedding for the target cell node v. Formally, this can be written as

$$h_v = \sigma\left(\sum_{u\in\mathcal{N}(v)} \alpha_{v,u}\mathrm{MLP}^a(h_v^i, h_u)\right) \quad v \in V_c. \tag{6}$$

Here, σ represents the ReLU activation function, and V_c denotes the set of all cell nodes. The above attention-based approach allows our GNN model to adeptly capture and prioritize the most impactful predecessor information. This mechanism is particularly effective in mimicking the delay calculation process in circuit design, ensuring that our model can properly capture the basic timing condition.

On the other hand, the message passing scheme along net edges represents a one-to-multiple relationship, as each net consists of one single driver pin and several sink pins. Similarly, the embedding of a net node v consists of two parts: 1) the received message from the driver node and 2) the intranet information (net features). Formally, the computation of the embedding of net node v can be written as

$$h_v = \mathrm{MLP}^n\big(h_d, h_{d,v}^i\big) \quad v \in V_n. \tag{7}$$

Here, V_n denotes the set of net nodes, h_d represents the embedding of the driver pin node of v, and $h_{d,v}^i$ is the initial net features of the edge (d, v). Since the cell nodes and net nodes alternate in the propagation flow, the above two message aggregating schemes will take place alternatively.

An important property of general GNNs is the permutation invariance to vertex order, ensuring consistent network output regardless of vertex order. This property is essential for the GNN's ability to generalize effectively across various graph structures. In the case of our customized GNN, which employs a predefined topological order, it is important to note that the topological order of a DAG is unique and deterministic.

V. Layout Information Extraction

Netlist information alone is insufficient to accurately model the complex timing behavior during timing optimization. To boost the model performance, we incorporate additional information from the layout to capture the impact of timing optimization. In this section, we propose a method for generating endpoint-wise layout embeddings using a U-Net model and a novel layout masking technique. The workflow is shown in Fig. 4.

A. Optimization-Related Layout Features

Our layout embedding flow initiates with the extraction of layout features, which are subsequently utilized as inputs for our CNN model. These input layout features are selected for their strong correlation with timing optimization objectives. The selection of input features is guided by two key observations.

1) Most timing optimization strategies require additional space. Therefore, denser layouts often provide fewer opportunities for timing optimization.
2) Regions occupied by macro cells are generally unsuitable for timing optimization due to their fixed nature.

Based on these insights and the success of considering routing congestion as a feature in previous studies [41], [42], we have chosen three distinct features to capture the essential aspects of the layout.

1) *Cell Density Map:* An evaluation of the density (number) of cells within each bin.
2) *Rectangular Uniform Wire Density (RUDY [41]) Map:* An estimation of wiring demand, which reflects the wire density and congestion condition within each bin.
3) *Macro-cell region map* represents the layout area occupied by macro cells.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.

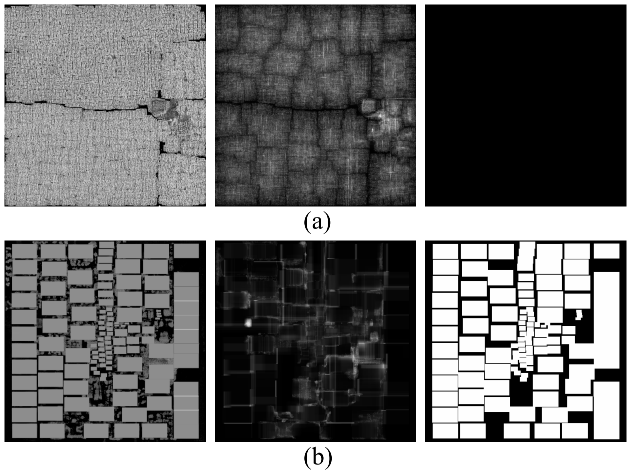

Fig. 5. Layout feature maps: cell density, RUDY, and macro cells region (from left to right). (a) Or1200. (b) Rocket.

These features are pivotal in assessing the layout's potential for timing optimization.

To generalize the layout features from circuit designs with diverse scales, we divide the overall layout into $M \times N$ bins (we set both M and N as 512). Here, we use a square configuration to align with the standard input shape requirements of CNNs. For rectangular layouts, we transform them into a square format by adjusting the bin dimensions. All three feature maps are derived directly from the placed chip layout. These stacked feature maps are then fed into our CNN model, as shown in Fig. 4.

To illustrate how these three feature maps vary across different circuit designs, Fig. 5 shows the cell density map, RUDY map, and macro map for two designs. Fig. 5(a) shows the layout feature maps for Or1200 CPU core while Fig. 5(b) illustrates the feature maps for the Rocket SoC design. The layout information clearly differs between the designs.

B. CNN Architecture

Regarding the CNN model, we adopt the U-net [43] architecture, which has been highly successful in semantic segmentation. Compared to conventional CNNs, U-Net incorporates multiscale contextual information, generating more expressive layout embeddings.

As shown in Fig. 4, the U-net model adopts the encoder-decoder structure. The encoder progressively reduces the dimensionality of the input image by applying a series of downsampling blocks. This process effectively captures the underlying features at various levels of abstraction. The decoder, on the other hand, consists of multiple upsampling units that gradually restore the compacted feature map to the original input size.

In practice, we adopt the ResNet [44] backbone architecture for the encoding path, consisting of three downsampling blocks with skip connections. The downsampling blocks reduce the spatial dimensions of the feature maps by processing the input through two 3×3 convolutional layers. The first convolutional layer employs a stride of 2, effectively reducing the input dimensions by half. Concurrently, the skip connections [43] in U-Net retain spatial features that are lost during downsampling. As shown in Fig. 4, the output from the downsampling path is concatenated with the corresponding output from the upsampling path of the same level. Following the encoding path, the bottleneck block refines the output from the encoder using 3×3 convolutions. The decoder then commences the process of reconstructing the compressed feature map to its original dimensions. This is achieved through three upsampling blocks, each doubling the size of its input using a transposed convolutional layer. Additionally, each upsampling block merges the corresponding feature map from the downsampling block at the same level via skip connections. This combination is then further processed with 3×3 convolutions.

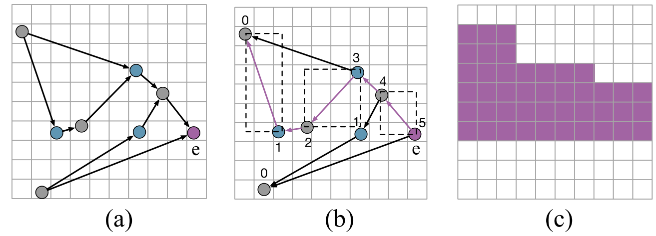

Fig. 6. Example of our proposed masking technique applied to an endpoint e, where purple, blue, and gray represent the endpoint, net nodes, and cell nodes, respectively. The number next to each node in (b) indicates its topological level, and the purple lines depict the longest path P_e for e. The dotted boxes illustrate the critical region $\mathbf{R}_e$, which consists of edge bounding boxes along P_e. (a) Input graph. (b) Finding path. (c) Generating mask.

C. Endpoint-Wise Masking

After processing through the U-net model, the input layout features are transformed into a fused map representing the overall layout condition. This fused map can then be fed into an FCN, producing a low-dimensional layout embedding vector. However, this straightforward approach results in all endpoints sharing the same layout information, ignoring the varying impact that timing optimization might have on different endpoints. Notably, the optimizer typically prioritizes critical endpoints with negative slack, resulting in a more pronounced effect on these endpoints compared to noncritical ones.

To address the above limitation, we propose a novel masking strategy to extract distinct, endpoint-specific layout information. Specifically, the layout embedding of an endpoint e aggregates information only from its most relevant region, termed as the critical region of e. This method is motivated by the close relationship between the arrival time at e and the longest path from PIs to e. Consequently, we determine the critical region based on this longest path, ensuring more targeted and effective layout information is extracted for each endpoint. Finding the longest paths in graphs is well studied [45], [46]. Here, we adopt a simple, yet effective way based on depth-first search (DFS).

Fig. 6 illustrates our proposed endpoint-specific masking technique. To find the longest path(s) for each endpoint, we first associate each node with its topological level.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.

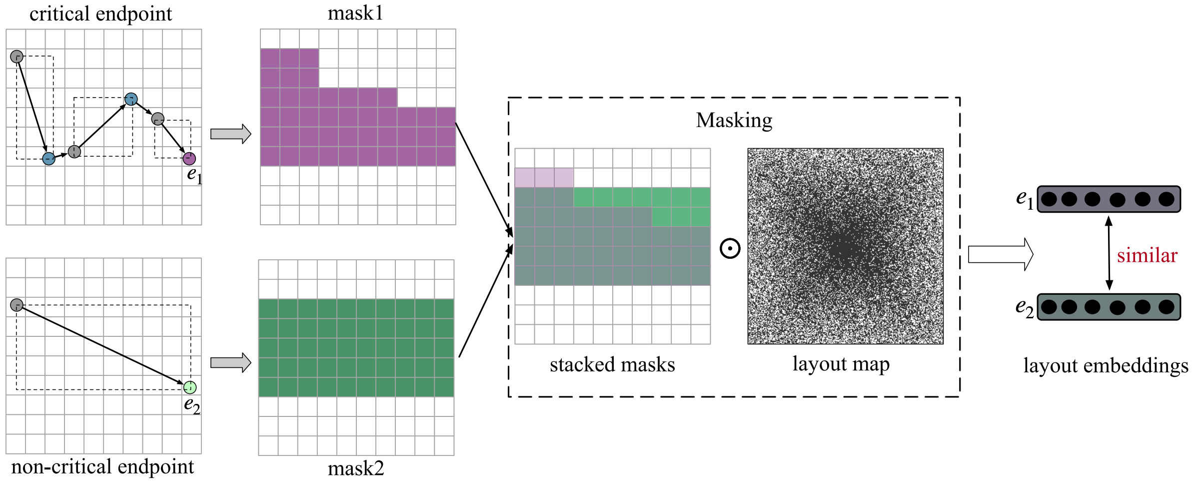

Fig. 7. Example to illustrate the drawback of layout masking without adaptive adjustment. Suppose the endpoint e_1 is critical while e_2 is noncritical. Since their layout mask covers a similar region, their masked layout embeddings are similar, which contradicts to their criticality.

Subsequently, we reverse the graph's edge direction and perform a DFS starting from each endpoint e to trace its longest path. During the DFS, there may be several candidate branches (successor nodes), and visiting all branches is computationally costly. Fortunately, we have an effective strategy for pruning candidate branches and selecting the target successor node. Since the topological level i of a node v^i implies the distance from PIs to it, only the successor node(s) with topological level $i-1$ are on the longest path(s) from PIs to v^i. Based on this, we can quickly and safely choose the successor node at each step. The DFS process concludes when it reaches a PI node, with the visited nodes forming the longest path from the PIs to endpoint e. Let m be the number of longest paths for a given endpoint, and l be the maximum length of these paths. Consequently, the computational complexity for processing each endpoint is $O(m \times l)$, which approximates to $O(1)$. Additionally, this method can be executed concurrently across various endpoints, significantly enhancing its speed. While the above method offers a quicker alternative to standard timing analysis for identifying critical paths, it sacrifices some accuracy and provides only a rough estimate. In future studies, we plan to explore more precise methods for identifying critical paths, including the application of traditional timing analysis techniques.

After determining the longest path P_e for each endpoint e, we create e's critical region $\mathbf{R}e$ by taking the union regions covered by the bounding boxes of the edges along P_e. To be specific, the bounding box for an edge (d, s) is the rectangular area defined by the coordinates of the driver node d and sink node s, as shown in

$$\begin{aligned}\mathbf{B}_{d,s} = [&\min \quad \{x_d, x_s\}, \max \quad \{x_d, x_s\}] \\ &\times \left[\min \quad \{y_d, y_s\}, \max \quad \{y_d, y_s\}\right].\end{aligned} \tag{8}$$

Here, $\times$ denotes the Cartesian product, and (x_d, y_d) and (x_s, y_s) are the positions of drive node d and sink node s, respectively. Furthermore, the critical region $\mathbf{R}_e$ can be formulated as

$$\mathbf{R}_e = \bigcup_{\{d,s\} \in E^n(P_e)} \mathbf{B}_{d,s}. \tag{9}$$

Here, P_e is e's longest path and $E^n(P_e)$ denotes the net edges along P_e. The critical mask $\mathbf{M}^e$ is then constructed based on R_e as $M^e_{ij} = 1 \ \forall (i, j) \in \mathbf{R}_e$. With the region mask $\mathbf{M}^e$, the unique layout map $\mathbf{M}^L_e$ for endpoint e can be calculated as

$$\mathbf{M}^L_e = \mathbf{M}^e \odot \mathbf{M}^L. \tag{10}$$

Here, $\odot$ denotes the Hadamard product, and M^L denotes the fused layout map produced by the U-net model. Then, a shared FCN is used to convert the endpoint-wise layout maps to low-dimensional layout embeddings, as shown in Fig. 4.

VI. Adaptive Layout Mask Adjustment

Our proposed masking technique focuses on the most relevant layout region to each endpoint, effectively extracting unique, endpoint-wise layout information. With the generated layout masks, our framework successfully captures the individual impact of global layout information on each endpoint.

However, it is important to note that the layout embeddings produced by this method are not flawless. The proposed masking technique implicitly assumes that the layout information within the same region has an equal impact on different endpoints. Unfortunately, this assumption does not always prove to be accurate.

Fig. 7 illustrates an example with layout masks for two endpoint e_1 and e_2. Let us assume that e_1 is a critical endpoint, meaning that its slack is negative, while e_2 is noncritical. Intuitively, the impact of timing optimization/layout information on e_2 should be less significant than that on e_1, given that timing optimization prioritizes the critical timing endpoints or paths. This suggests that the layout embedding of e_1 should have a more pronounced influence on the final fuse embedding than that of e_2. However, the masking technique proposed in Section V-C could result in similar layout embeddings for these two endpoints, given that their critical region masks overlap significantly, as shown in Fig. 7. This overlap can distort the impact of timing optimization on different endpoints in the final fuse embeddings, thus limiting the effectiveness of multimodal fusion.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.

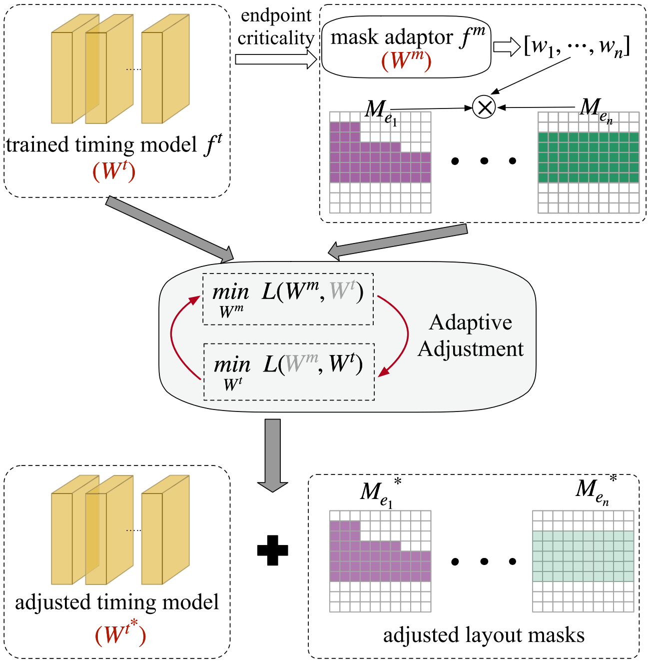

Fig. 8. Flow of our adaptive mask adjustment process. The color saturation of the mask reflects their timing criticality; the higher, the more critical. The timing model f^t and the mask adaptor f^m are parameterized by W^t and W^m, respectively. $\times$ represents the multiplication operator, and we use the gray color to denote whether the learnable parameters W^t/W^m are frozen or not.

The key to overcoming this drawback is distinguishing between the layout masks of timing endpoints that overlap similar regions, particularly those with differing timing criticality. Based on the fact that timing optimization is more focused on the critical endpoints, our strategy should focus on amplifying the layout embedding of critical endpoints while downplaying those of noncritical endpoints. In this section, we introduce a scheme that learns to adaptively adjust each endpoint's layout mask to leverage layout information more effectively.

Fig. 8 shows the flow of our adaptive mask adjustment scheme, which takes a trained timing model as input. The process begins by evaluating the criticality of each timing endpoint using the trained timing model, which can be written as follows:

$$c_e = -\hat{\text{slack}}_e = -\left(r_e - \hat{a}_e\right) \tag{11}$$

where c_e gives the criticality of endpoint e, r_e is the required time of e, and $\hat{a}_e$ is the arrival time of e predicted by the timing model. The weight of each timing endpoint is then initialized based on its evaluated criticality using a mask adaptor f^m, which can be written as follows:

$$w_e = \sigma\left(f^m(c_e, \vec{t}_e)\right) \tag{12}$$

where $\vec{t}_e$ is a one-hot vector that denotes the type of the endpoint. Here, we classify the endpoints into three types: 1) data input pin; 2) reset/set pin; or 3) enable pin. σ represents the sigmoid activation function, and f^m is instantiated as a MLP model. The weights are then multiplied with the layout masks of the timing endpoints, resulting in adjusted layout embeddings.

Algorithm 1 Adaptive Mask Adjustment

for $k = 1, 2, 3 \cdots$ **do**
  while not converged **do** ▷ Mask Adaptor Iteration
    $W^m \leftarrow \text{argmin}_{W^m} \mathcal{L}(W^m, W^t)$; ▷ Adam
  end while
  while not converged **do** ▷ Timing Model Iteration
    $W^t \leftarrow \text{argmin}_{W^t} \mathcal{L}(W^m, W^t)$; ▷ Adam
  end while
end for
return W^t, W^m;

Following this, we apply an iterative optimization process to adaptively adjust both the mask adaptor and the timing model. The optimization target is to minimize the MSE between the ground truth and predicted endpoint arrival time, which can be written as

$$\begin{aligned}\mathcal{L} &= \frac{1}{n}\sum_{i=1}^{n}\left(a_i - \hat{a}_i\right) \\ &= \frac{1}{n}\sum_{i=1}^{n}\Big(a_i - f^r\big(f^n(\mathcal{N}, e_i),\ \big(\sigma(f^m(c_i, \vec{t}_e)) \\ &\qquad \times \mathbf{M}_{e_i}\big) \odot f^l(\mathcal{M})\Big)^2\end{aligned} \tag{13}$$

where e_i denotes the ith endpoint, a_i and $\hat{a}_i$ denotes the ground truth and predicted arrival time of e_i, c_i gives the criticality of e_i, and $\mathbf{M}_{e_i}$ is the layout mask of e_i. f^r denotes the modality-fusion model, f^n represents the netlist embedding model (the GNN), and f^l represents the layout embedding model (the CNN). $\mathcal{N}$ denotes the input netlist, and $\mathcal{M}$ represents the input layout features. Let us denote the learnable parameters of the timing model as W^t, which include f^r, f^n, and f^l, and the parameters of the mask adaptor f^m as W^m. Then, the optimization target can be written as

$$\min_{W^m, W^t} \mathcal{L}\left(W^m, W^t, \mathcal{N}, \mathcal{M}\right). \tag{14}$$

As shown in Algorithm 1, we apply a bi-level optimization scheme that alternatively adjusts the timing model and the layout mask adaptor. In the first level, we freeze the timing model and focus solely on adjusting the mask weights. Conversely, in the second level, we freeze the mask adaptor and focus on optimizing the timing model. This two-level optimization occurs iteratively until either convergence is achieved or a predetermined number of epochs have passed. It is important to note that we do not directly optimize the timing model and the layout mask adaptor simultaneously, as this straightforward approach has proven ineffective, possibly due to the complexity of optimizing these two targets concurrently.

The above scheme learns a layout mask adaptor that assigns an appropriate weight for each timing endpoint, resulting in distinct layout masks (or embeddings) for endpoints with significant critical region overlap but differing timing criticality. Consequently, the individual impact of layout information on each timing endpoint is accurately reflected, leading to more precise predictions.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.

TABLE II

COMPARISON ON THE TEST BENCHMARKS. THE LOCAL DELAY PREDICTION RESULTS OF THE BASELINE MODELS ARE PRESENTED IN THE LEFT COLUMNS, WHICH ARE CALCULATED UPON THE UNREPLACED NETS/CELLS. NOTE THAT [4] AND [5] INCORPORATE DRIVER CELL DELAY AND NET DELAY, WHILE [6] PREDICTS THEM SEPARATELY. THE MODELS' PERFORMANCE ON GLOBAL TIMING METRICS, I.E., ENDPOINT ARRIVAL TIME, IS LISTED IN THE RIGHT COLUMNS, WHERE THE BEST RESULTS ARE HIGHLIGHTED IN BOLDFACE. OUR MODEL OUTPERFORMS THE PREVIOUS WORKS ON ALL THE BENCHMARKS, DEMONSTRATING ITS GREAT SUPERIORITY

Benchmark	baselines' net/cell delay prediction (R^2 score)			Endpoint arrival time prediction (R^2 score)				
	DAC19	DAC22-he	DAC22-guo	DAC19 [4]	DAC22-he [5]	DAC22-guo [6]	DAC23-wang [47]	PRO-TIME
arm9	0.0101	-0.5187	-0.2960 / -1.8234	0.6655	0.7304	0.8279	0.8852	**0.9152**
chacha	-0.1389	-0.1008	-0.0813 / -0.2737	0.4406	0.6146	-0.0253	0.9027	**0.9374**
hwacha	0.0519	-0.0323	-0.8003 / -0.8630	0.2752	0.5186	0.7090	0.8623	**0.8908**
or1200	-0.0395	-0.3051	-3.5679 / -0.0924	0.3226	0.4484	0.6776	0.8081	**0.8375**
sha3	0.3941	0.5554	-0.3713 / 0.1230	0.7784	0.7917	0.8464	0.9035	**0.9221**
avg	0.0555	-0.0803	-1.0234 / -0.5859	0.4965	0.6207	0.6071	0.8724	**0.9004**

VII. EXPERIMENTS

A. Experimental Setup

We developed the framework with DGL [48] and PyTorch [49]. The neural networks were trained and evaluated on a Linux machine with 16 Intel Xeon Gold 6226R cores (2.90 GHz), one GeForce RTX 3090 Ti graphics card, and 24 GB of main memory. For our model's hyperparameters, we choose three-layer MLPs with a hidden layer dimension of 256 for the GNN part. The CNN input size is $3 \times 512 \times 512$, and the network architecture is illustrated in Fig. 4. Both the endpoint-wise netlist and layout embedding dimension are 128, and the MLP used for regression has three layers with a hidden dimension of 512. Our model was trained with a learning rate of 0.001 and a batch size of 1024 for 100 epochs. After that, the layout mask adjustment scheme was run for ten iterations (20 epochs) using the training dataset.

A dataset of ten open-source designs is prepared, as listed in Table I. We obtain the designs from Chipyard open-source projects and other resources on GitHub. Almost all the designs are RISC-V supported except chacha. We randomly divide the dataset into five cases for training and five cases for testing. For dataset generation, we apply Cadence Genus with the edge-cut 7-nm ASAP7 PDK [50] for synthesis, and Cadence Innovus for placement, timing optimization, and routing.

For evaluation, we use the coefficient of determination (R^2) as the metric. R^2 measures how well the model fits the data by assessing the proportion of variance in the dependent variable explained by the independent variables. Formally, the computation of R^2 can be written as

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}. \tag{15}$$

Here, n denotes the number of samples, $\bar{y} = (\sum_{i=1}^{n} y_i/n)$ gives the mean of the ground-truth values, and y_i and $\hat{y}_i$ represent the ground-truth and predicted values of the ith sample, respectively. The best possible R^2 score is 1, which indicates a perfect fit of the model to the data. On the other hand, the worst possible R^2 score can be negative, which happens when the model does not follow the trend of the data, so it fits worse than a horizontal line.

B. Overall Performance

We first compare our method with previous state-of-the-art works [4], [5], [6] for prerouting timing evaluation, which rely on local net/cell delay prediction. As previously stated, netlist restructuring during timing optimization poses a challenge for these works. Since restructured subnetlists cannot be labeled, we adapt the baseline models to our task in a semi-supervised fashion, training with unchanged cells/nets/pins. The previous two-stage methods [4], [5] are supervised by local net/cell delay, the prior end-to-end method [6] is supervised by net/cell delay, pin slew, and pin arrival time, and our model is supervised by endpoint arrival time. Furthermore, we also compare our work with DAC 2023 [47]. All the baseline models are well-trained on our training dataset using the official settings in their papers. To ensure a fair comparison, we used identical benchmark settings (library, timing targets, and synthesis process) across all baseline methods.

We summarize our findings from the results in Table II as follows.

1) Our proposed framework vastly outperforms all the previous baseline approaches on all benchmarks for sign-off global timing metrics prediction.
2) Previous methods show relatively low performance on local net/cell delay prediction, demonstrating the difficulty of modeling the influence of timing optimization locally with only prerouting information.
3) Prediction performance on local net/cell delay is inconsistent with that on global timing metrics, i.e., a higher R^2 score on local delay leads to a lower R^2 score on endpoint arrival time. This might be due to the aforementioned feature mismatching.
4) Our global-view approach (supervised only by global timing information) outperforms prior local-view fashion when timing optimization is taken into account.
5) A performance gain of 2.8% is achieved compared to our previous conference version, which demonstrates the efficacy of our new techniques. Fig. 9 visually compares our new flow with its previous version, showing that the predicted arrival time of our new flow aligns better with the ground-truth values.

C. Ablation Study

We conducted ablation studies to further assess the efficacy of the proposed techniques, with the configurations detailed in Table III. Fig. 10 shows the average performance of the models on test designs, and our observations are summarized as follows.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.

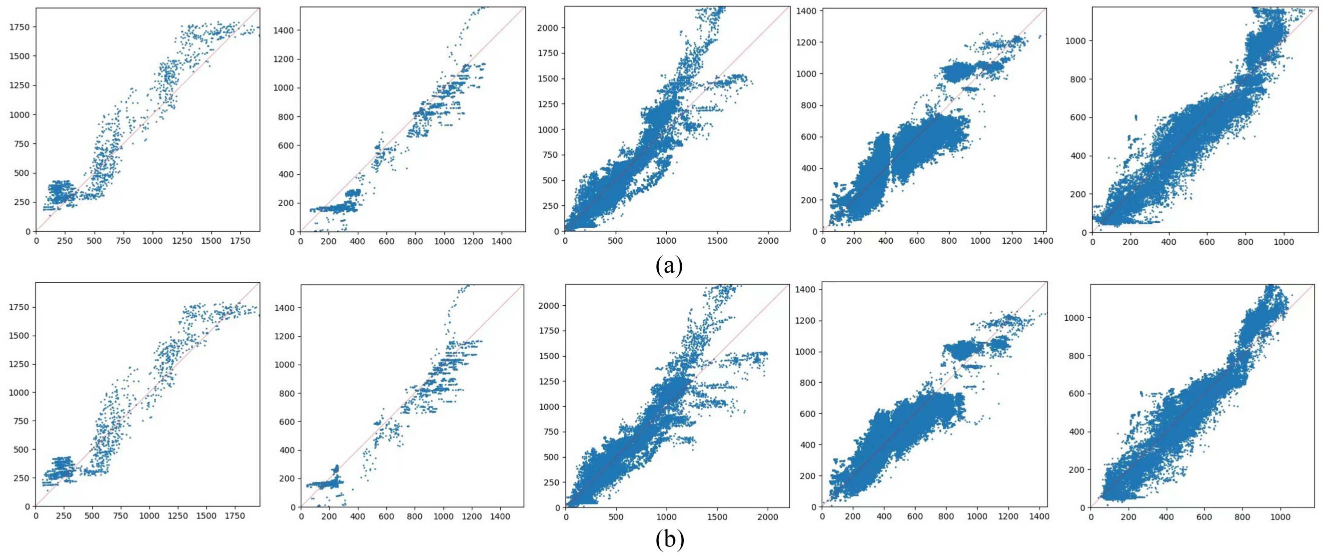

Fig. 9. Visualization of the coefficients between the predicted and ground-truth arrival time, where the *x*-axis is for the predicted values and the *y*-axis is for the ground-truth values. From left to right, the figures show the results for arm9, chacha, hwacha, or1200, and sha3. (a) Results of DAC 2023 [47]. (b) Results of the proposed PRO-TIME.

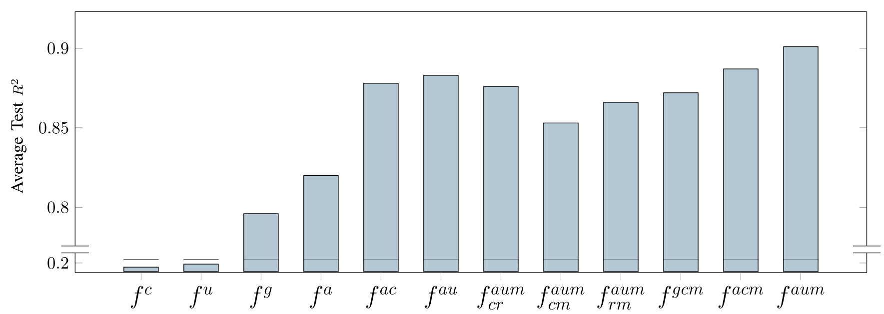

Fig. 10. Results of the ablation study. Here, *g* represents our customized GNN without attention, *a* represents our customized GNN with attention, *c* represents a typical CNN model, *u* denotes the U-net model, and *m* denotes our adaptive mask adjustment scheme.

1) Single-modal models f^g, f^a, f^c, and f^u, which rely solely on netlist or layout information, demonstrated suboptimal performance. This outcome underscores the benefits of multimodal fusion in prerouting timing prediction.
2) Comparison between f^g and f^a reveals an enhancement in the single-modal model's performance when incorporating our attention mechanism. This improvement suggests that our attention mechanism effectively enhances the extraction of precise timing information from netlists.
3) When contrasting f^c with f^u, and f^{ac} with f^{au}, as well as f^{aum} with f^{acm}, it indicates that the U-net model outperforms the conventional CNN model in processing layout information. Notably, our adaptive mask adjustment method shows a more significant improvement when paired with U-net compared to when used with CNN.
4) Analysis of $f^{\text{aum}}cr, f^{\text{aum}}cm, f^{\text{aum}}_{rm}$, and f^{aum} indicates that omitting any layout feature map significantly reduces performance, with the RUDY map having the most substantial impact.
5) Our adaptive mask adjustment scheme brings a performance gain of more than 2%. Moreover, this approach proves to be more effective than merely using estimated slack as a feature (as shown in Fig. 11), highlighting the advantage of our method in refining model performance by providing more nuanced control over feature representation.

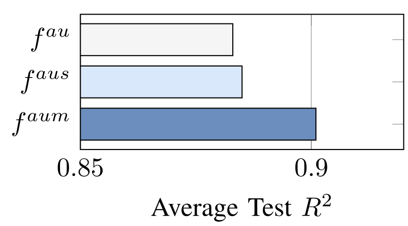

Fig. 11. Comparison between model without refinement (f^{au}), model with our adaptive mask adjustment (f^{aum}), and model that directly uses the estimated slack as an additional feature (f^{aus}).

D. Runtime Analysis

In this part, we show the speedup of our model compared to the industry-leading commercial tool. The timing optimization and routing procedure are extremely time-consuming in modern VLSI design flow due to their high complexity. In contrast,

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.

TABLE III
CONFIGURATION FOR ABLATION STUDY. GAT REFERS TO ATTENTION-BASED GNN, AMA REPRESENTS ADAPTIVE MASK ADJUSTMENT, AND SF REPRESENTS THE METHOD THAT DIRECTLY USES THE ESTIMATED SLACK AS A FEATURE. C-R-M REPRESENTS THE USAGE OF THREE LAYOUT FEATURES: CELL DENSITY, RUDY, AND MACRO CELL REGION

Model	Configuration					
	Netlist		Layout			Refine
	GNN	GAT	CNN	U-net	feature	AMA
f^c			✓		C-R-M	
f^u				✓	C-R-M	
f^g	✓					
f^a		✓				
f^{ac}		✓	✓		C-R-M	
f^{acm}		✓	✓		C-R-M	✓
f^{gcm}	✓		✓		C-R-M	✓
f^{au}		✓		✓	C-R-M	
f^{aum-cr}		✓		✓	C-R	✓
f^{aum-cm}		✓		✓	C-M	✓
f^{aum-rm}		✓		✓	R-M	✓
f^{aum}		✓		✓	C-R-M	✓

TABLE IV
RUNTIME (S) COMPARISON WITH AN INDUSTRY-LEADING COMMERCIAL TOOL WHERE OPT IS FOR OPTIMIZATION, STA IS FOR STATIC TIMING ANALYSIS, PRE IS FOR PREPROCESSING, AND INFER IS FOR INFERENCE. WE CAN ACHIEVE AN OVER 3000× SPEEDUP ON AVERAGE

design	commercial (20 threads)				PRO-TIME			
	opt	route	sta	total	pre	infer	total	speedup
jpeg	7863	624922	227	633012	20.63	11.42	32.05	**19751**×
rocket	16239	19161	167	35567	18.53	3.98	22.51	**1580**×
smallboom	9051	53942	152	63145	19.72	10.03	29.75	**2123**×
steelcore	1294	747	20	2061	0.39	2.44	2.83	**728**×
xgate	338	630	17	985	0.34	1.25	1.59	**619**×
arm9	305	1825	16	2146	0.88	4.12	4.00	**430**×
chacha	1621	1794	23	3438	0.82	2.67	3.49	**985**×
hwacha	43883	136946	241	181070	23.89	11.88	35.77	**5062**×
or1200	28641	40291	339	69271	112.20	13.52	125.72	**551**×
sha3	18785	16870	185	35840	24.95	5.46	30.41	**1179**×
avg.	12802	89713	139	102654	22.23	6.68	28.91	**3297**×

ML models have overwhelming superiority in efficiency. The results are listed in Table IV, where the preprocessing stage of our model includes graph construction, topological level generation, and endpoint-wise critical region generation. We find that our proposed flow runs 3297 times faster on average than the commercial tool to evaluate the sign-off global timing metrics at a small cost of around 10% R^2 score loss. In a word, our proposed model successfully performs fast and accurate optimization-aware prerouting timing evaluation.

VIII. CONCLUSION

Prior studies have noted the importance of fast and accurate prerouting timing prediction in reducing design cycles. However, modern ML-assisted works following a local-view fashion did not consider the impact of timing optimization, leading to performance degradation in real-world applications. This article has presented a novel endpoint embedding framework with multimodal fusion by utilizing both GNN and CNN to incorporate netlist and layout information. An end-to-end flow is further developed for prerouting prediction on post-routing global timing metrics, with the impact of a modern timing optimizer taken into account. The experimental results on advanced 7-nm RISC-V designs highlight the importance of multimodal fusion in modeling the impact of timing optimization. Future work will expand upon this research by developing a more advanced net/cell delay model. One approach involves a sophisticated net delay estimator that considers low, medium, and high fanout nets, rather than solely relying on Manhattan distance. Additionally, we plan to use our method's predictions to enhance prerouting timing optimization.

REFERENCES

[1] M. Vujkovic, D. Wadkins, B. Swartz, and C. Sechen, "Efficient timing closure without timing driven placement and routing," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2004, pp. 268–273.

[2] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal delay in RC tree networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 2, no. 3, pp. 202–211, Jul. 1983.

[3] A. I. Abou-Seido, B. Nowak, and C. Chu, "Fitted Elmore delay: A simple and accurate interconnect delay model," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 12, no. 7, pp. 691–696, Jul. 2004.

[4] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[5] X. He, Z. Fu, Y. Wang, C. C. Liu, and Y. Guo, "Accurate timing prediction at placement stage with look-ahead RC network," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 1213–1218.

[6] 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 *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 1207–1212.

[7] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single pert-like traversal," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2003, pp. 621–625.

[8] C. S. William Swartz, "Timing driven placement for large standard cell circuits," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 1995, pp. 211–215.

[9] I. S. Bustany, D. Chinnery, J. R. Shinnerl, and V. Yutsis, "ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, 2015, pp. 157–164.

[10] A. Srivastava, D. Sylvester, and D. Blaauw, *Statistical Analysis and Optimization for VLSI: Timing and Power*, vol. 59. New York, NY, USA: Springer, 2005.

[11] D. Sinha, N. V. Shenoy, and H. Zhou, "Statistical gate sizing for timing yield optimization," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2005, pp. 1037–1041.

[12] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Dordrecht, The Netherlands: Springer, 2011.

[13] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 1024–1034.

[14] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–17.

[15] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proc. ACM Int. Conf. Knowl. Disc. Data Min. (KDD)*, 2019, pp. 793–803.

[16] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[17] J. Zhou et al., "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, Apr. 2021.

[18] A. Mirhoseini et al., "A graph placement methodology for fast chip design," *Nature*, vol. 594, pp. 207–212, Jun. 2021.

[19] Y. Ma et al., "High performance graph convolutional networks with applications in testability analysis," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.

[20] Z. Wang et al., "Functionality matters in netlist representation learning," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 61–66.

[21] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2012, pp. 3642–3649.

[22] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 3431–3440.

[23] J. Fu, H. Zheng, and T. Mei, "Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 4476–4484.

[24] C. Yu, H. Xiao, and G. De Micheli, "Developing synthesis flows without human knowledge," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2018, pp. 1–6.

[25] Z. Xie et al., "PowerNet: Transferable dynamic IR drop estimation via maximum convolutional neural network," in *Proc. IEEE/ACM Asia–South Pacific Design Autom. Conf. (ASPDAC)*, 2020, pp. 13–18.

[26] V. A. Chhabria, A. B. Kahng, M. Kim, U. Mallappa, S. S. Sapatnekar, and B. Xu, "Template-based PDN synthesis in floorplan and placement using classifier and CNN techniques," in *Proc. IEEE/ACM Asia–South Pacific Design Autom. Conf. (ASPDAC)*, 2020, pp. 44–49.

[27] Z. Xie et al., "RouteNet: Routability prediction for mixed-size designs using convolutional neural network," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–8.

[28] J. Chen, J. Kuang, G. Zhao, D. J.-H. Huang, and E. F. Young, "PROS: A plug-in for routability optimization applied in the state-of-the-art commercial EDA tool using deep learning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–8.

[29] S. Liu, Q. Sun, P. Liao, Y. Lin, and B. Yu, "Global placement with deep learning-enabled explicit routability optimization," in *Proc. IEEE/ACM Proc. Design, Autom. Test Eurpoe (DATE)*, 2021, pp. 1821–1824.

[30] R. Chen et al., "Faster region-based hotspot detection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 3, pp. 669–680, Mar. 2022.

[31] H. Yang, L. Luo, J. Su, C. Lin, and B. Yu, "Imbalance aware lithography hotspot detection: A deep learning approach," *J. Micro/Nanolithogr. MEMS, MOEMS*, vol. 16, no. 3, 2017, Art. no. 33504.

[32] H. Yang et al., "VLSI mask optimization: From shallow to deep learning," *Integration*, vol. 77, pp. 96–103, Mar. 2021.

[33] B. Jiang, L. Liu, Y. Ma, H. Zhang, B. Yu, and E. F. Young, "Neural-ILT: Migrating ILT to neural networks for mask printability and complexity co-optimization," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–9.

[34] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 2, pp. 423–443, Feb. 2019.

[35] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2011, pp. 689–696.

[36] H. Ren, S. Nath, Y. Zhang, H. Chen, and M. Liu, "Why are graph neural networks effective for EDA problems?" in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2022, pp. 1–8.

[37] Z. Wang, Z. He, C. Bai, H. Yang, and B. Yu, "Efficient arithmetic block identification with graph learning and network-flow," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 8, pp. 2591–2603, Aug. 2023.

[38] S. Zheng, L. Zou, P. Xu, S. Liu, B. Yu, and M. Wong, "Lay-Net: Grafting netlist knowledge on layout-based congestion prediction," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2023, pp. 1–9.

[39] A. Vaswani, "Attention is all you need," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 1–21.

[40] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–11.

[41] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *Proc. IEEE/ACM Proc. Design, Autom. Test Eurpoe (DATE)*, 2007, pp. 1–6.

[42] V. A. Chhabria, W. Jiang, A. B. Kahng, and S. S. Sapatnekar, "A machine learning approach to improving timing consistency between global route and detailed route," *ACM Trans. Design Autom. Electron. Syst.*, vol. 29, no. 1, pp. 1–25, 2023.

[43] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervention (MICCAI)*, 2015, pp. 234–241.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778.

[45] D. Karger, R. Motwani, and G. D. Ramkumar, "On approximating the longest path in a graph," *Algorithmica*, vol. 18, no. 1, pp. 82–98, 1997.

[46] R. Uehara and Y. Uno, "Efficient algorithms for the longest path problem," in *Proc. Int. Symp. Algorithms Comput.*, 2004, pp. 871–883.

[47] Z. Wang, S. Liu, Y. Pu, S. Chen, T.-Y. Ho, and B. Yu, "Restructure-tolerant timing prediction via multimodal fusion," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.

[48] M. Wang et al., "Deep graph library: Towards efficient and scalable deep learning on graphs," 2019, *arXiv:1909.01315*.

[49] A. Paszke et al., "Automatic differentiation in PyTorch," in *Proc. NIPS*, 2017, pp. 1–4.

[50] L. T. Clark et al., "ASAP7: A 7-nm finFET predictive process design kit," *Microelectron. J.*, vol. 53, pp. 105–115, Jul. 2016.

Ziyi Wang received the B.S. degree from the Department of Computer Science and Technology, Fudan University, Shanghai, China, in 2021. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His research interests include graph learning applications in electronic design automation and logic synthesis.

Siting Liu received the B.S. degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2020. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Her research interests include physical synthesis, machine learning application, and GPU acceleration in VLSI CAD algorithms.

Ms. Liu is a recipient of the Best Paper Award from DATE 2022 and the Best Paper Award Nomination from DATE 2021.

Yuan Pu received the B.S. degree in computer science from The Chinese University of Hong Kong, Hong Kong, in 2022, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, supervised by Prof. B. Yu.

His research interest includes combinatorial algorithms/AI in EDA.

Song Chen (Member, IEEE) received the B.S. degree in computer science from Xi'an Jiaotong University, Xi'an, China, in 2000, and the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2005.

He served with the Graduate School of Information, Production, and Systems, Waseda University, Shinjuku, Japan, as a Research Associate from August 2005 to March 2009 and as an Assistant Professor from April 2009 to August 2012. He is currently an Associate Professor with the School of Microelectronics, University of Science and Technology of China, Hefei, China. His research interests include on-chip communication systems, compiling and design automation for AI chips, and near-/in-memory computing architecture.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.

Tsung-Yi Ho (Fellow, IEEE) received the Ph.D. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2005.

He is a Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. His research interests include several areas of computing and emerging technologies, especially in the design automation of microfluidic biochips.

Prof. Ho was a recipient of the Best Paper Award at the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems in 2015. He currently serves as the VP Conferences of IEEE CEDA, and the Executive Committee of ASP-DAC and ICCAD. He is a Distinguished Member of ACM.

Bei Yu (Senior Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Dr. Yu received 11 Best Paper Awards from ICCAD 2013, 2021, and 2024, IEEE TSM 2022, DATE 2022, ASPDAC 2012 and 2021, ICTAI 2019, Integration, the VLSI Journal in 2018, ISPD 2017, and SPIE Advanced Lithography Conference 2016, six ICCAD/ISPD contest awards, the IEEE CEDA Ernest S. Kuh Early Career Award in 2021, the DAC Under-40 Innovator Award in 2024, and the Hong Kong RGC Research Fellowship Scheme Award in 2024. He has served as the TPC Chair for the ACM/IEEE Workshop on Machine Learning for CAD in 2019, the International Symposium of EDA in 2025, and in many journal editorial boards and conference committees.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on December 26,2025 at 12:53:17 UTC from IEEE Xplore. Restrictions apply.