Restructure-Tolerant Timing Prediction via Multimodal Fusion

Ziyi Wang[1†], Siting Liu[1†], Yuan Pu[1], Song Chen[2], Tsung-Yi Ho[1], Bei Yu[1]
[1]Chinese University of Hong Kong [2] University of Science and Technology of China

***Abstract*—Fast and accurate pre-routing timing prediction is crucial in the very-large-scale integration (VLSI) design flow. Existing machine learning (ML)-assisted pre-routing timing evaluators neglect the impact of timing optimization, which may render their approaches impractical in real circuit design flows. To model the impact of timing optimization, we propose an endpoint embedding framework that integrates netlist-layout information via multimodal fusion. An end-to-end flow is further developed for pre-routing restructure-tolerant prediction on global timing metrics. Comprehensive experiments on large-scale RISC-V designs with advanced 7-nm technology node demonstrate the superiority of our model compared to the SOTA pre-routing timing evaluators.**

I. Introduction

The breakthrough in advanced technology nodes has sparked interest in the circuit design flow since 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. Repetitive placement and routing (PnR) are needed to meet the sign-off timing constraints, which is time-consuming. As a result, timing prediction based on the placement solution has drawn researchers' attention to save the abundant routing running time and provide quick feedback to optimize timing early.

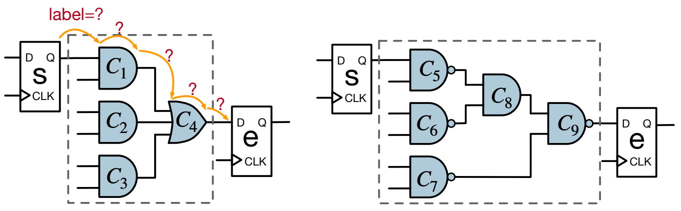
Efficient and accurate pre-routing timing prediction is necessary for timing-driven placement engines. In practice, the linear RC static timing analysis (STA) model, Elmore's model [1], is widely-used to get quick timing evaluation with only placement results. However, Elmore's model is imprecise due to inaccurate wire estimation without actual routing information.

Trending machine learning (ML) techniques have opened up new opportunities for pre-routing timing evaluation [2]–[4]. In general, these works follow a local-view fashion that relies on local net/cell delay prediction. They can be classified into two categories, two-stage or end-to-end. The two-stage approaches [2], [3] first predict local net/cell delays and then apply PERT traversals [5] to evaluate the global timing metrics, i.e., endpoint arrival time. On the other hand, the end-to-end method [4] can directly predict the global timing metrics in a single run. It generates and propagates net/cell embeddings in the topological order and then predicts at the endpoints. Nevertheless, [4] still relies on local net/cell delay prediction as auxiliary tasks. Overall, these studies highlight the need for pre-routing timing prediction and point out the success of ML-assisted methods.

Unfortunately, none of the above works considered timing optimization, i.e., they did not run a timing optimizer in their data-generating flow. Timing optimization is a necessary step in modern design flows to meet timing constraints, which is composed of various optimizing techniques. Many of these techniques, e.g., gate rewrite, irrevocably restructure the netlist, posing a severe challenge to prior local-view works. To be more specific, netlist restructuring causes a mismatch between local input features and ground-truth features in the restructured sub-regions, as shown in Fig. 1, which inevitably leads to inaccurate prediction and performance degradation. It is found

This work is supported by The Research Grants Council of Hong Kong SAR (No. CUHK14209420).

† Equal contributors



(a) Input netlist (b) Optimized netlist

Fig. 1 Example of netlist reconstruction after timing optimization. The sub-netlist within the dotted box is replaced to improve timing, which prohibits labeling the net/cell delays inside the box. Furthermore, it 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.

that on average 40% nets and 21% cells are replaced during timing optimization (details in TABLE I), indicating a significant influence on prior local-view works. Moreover, the unreplaced nets/cells are also greatly affected by timing optimization, as shown in TABLE I. Comparing flows with or without timing optimization, we find that timing optimization brings an average change of 59.6% to net delays and 33.3% to cell delays. The changes are calculated upon unreplaced nets/cells. The considerable impact on individual nets/cells further prevents ML-driven methods from correctly predicting local timing metrics when timing optimization is considered.

Based on the facts that timing endpoints are never replaced, and most commonly-used timing optimization techniques rely on global information and layout knowledge [6], we argue that a global endpoint-wise view from both netlist and layout is urgently needed for restructure-tolerant timing estimation, rather than the local-view fashion of previous works. This study focuses on the pre-routing prediction of sign-off global timing metrics, i.e., endpoint arrival time, considering the impact of the modern timing optimizer. Specifically, we propose a novel endpoint embedding framework that fuses layout and netlist information. An end-to-end flow is further developed for the restructure-tolerant global timing evaluation based on the learned endpoint embeddings.

The major contributions of this paper are listed as follows,

- To the best of our knowledge, we are the first to provide an end-to-end restructure-tolerant timing prediction flow.
- We develop an endpoint embedding framework that fuses layout-netlist information considering the impact of timing optimization.
- A customized graph neural network is presented to extract and aggregate endpoint-wise netlist information.
- A convolutional neural network with an endpoint-wise masking technique is developed to efficiently extract the unique layout information for each timing endpoint.
- We conduct experiments and ablation studies on large-scale RISC-V designs in the advanced 7-nm node to confirm the effectiveness of our proposed techniques.

The rest of the paper 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 describes extracting endpoint-wise netlist information. Section V introduces the proposed endpoint-wise layout information extraction technique. Section VI presents experimental results, followed by the conclusion in Section VII.

II. Preliminaries and Problem Definition

A. Timing Optimization

The tape-out circuit should not only satisfy the geometric constraints, e.g., non-overlapping cells but also meet the timing constraints. Modern physical design flows apply a timing optimization engine via a complex process to improve timing performance. As listed in TABLE I, timing optimization significantly impacts both local and global sign-off timing metrics. It is worth noting that though the overall optimizing direction is towards improving the global timing performance, i.e., total negative slacks of all the endpoints, its impact on an individual net/cell is uncertain. For instance, the delay of some nets might be hundreds of times larger than that generated by a flow without optimization, while others might be reduced to near zero. The uncertainty is caused by complicated global interplays between nets/cells [7], which makes predicting local net/cell delays extremely difficult considering the impact of timing optimization.

We divide timing optimization techniques into two classes: structure-preserved or structure-destructed. The structure-preserved techniques [8] improve timing while bringing no change to the netlist structure. A representative technique is gate-sizing [9] that chooses a better size for each gate from the cell library to optimize overall timing performance. On the other hand, structure-destructed techniques [10], e.g., Boolean restructuring, gate decomposition, etc., optimize timing by modifying the netlist structure. Such changes do not alter the circuit's functionality but can use additional gates or rewire the connections between existing gates to improve driving strength and signal integrity [10].

Fig. 1 gives an example of netlist reconstruction during timing optimization, where the multiple-input gates in the dotted box are replaced by more efficient gates. As can be seen, modification of the netlist structure brings considerable challenges to timing prediction. To begin, it prevents labeling sign-off delays for nets/cells within the replaced regions, e.g., C_1-C_4. As a result, prior local-view models can only be trained on the unchanged regions in a semi-supervised manner. Furthermore, it leads to feature mismatching in the replaced regions. To be specific, the input features used for delay prediction in the replaced sub-netlists, e.g., C_1-C_4's information, are mismatched with the ground-truth features, e.g., C_5-C_9's information. The mismatched features cause inaccurate local delay prediction in restructured regions, which further leads to an inconsistency between local delay supervision and global timing metrics prediction. In other words, the better the models fit on labeled (unreplaced) net/cell delays, the worse they fit on replaced regions and eventually on endpoint arrival time.

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 [6].

B. Graph and Convolutional Neural Network

Graph neural network (GNN) [11]–[13] has emerged as an appealing methodology for processing graph-structured data and mining graph information in recent years. It follows an iterative message-passing scheme to capture the structural information within nodes' neighborhoods. Let $G = \langle \mathcal{V}, \mathcal{E} \rangle$ denotes a graph, where $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$ is the vertex set, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set. Considering a K-layer GNN, the propagation of the k-th layer is represented as

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

where $\boldsymbol{a}_v^{(k)}$ and $\boldsymbol{h}_v^{(k)}$ denote the aggregated neighboring message and the embedding of vertex v at the k^{th} layer, respectively, and $\mathcal{N}(v)$ represents the set of neighboring nodes of v. AGGREGATE is a function that collects messages from a node's neighborhood, while COMBINE is used to integrate the node's previous embedding with the message from its neighborhood.

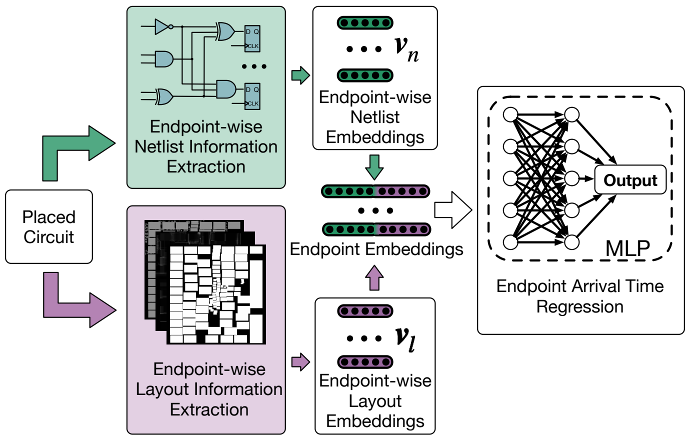
On the other hand, Convolutional neural network (CNN) is one of the most important deep learning models for retrieving local-global information of Euclidean data (grid-pattern data) [14]–[16]. It is built by a stack of locally connected layers, such as the linear convolution layers with non-linear activation operations, down-sampling pooling layers, and fully connected layers.

C. Problem Definition

Problem 1 (Restructure-tolerant timing prediction)**.** *Given the pre-routing layout and netlist of a design, our goal is to make an accurate and efficient estimation of the sign-off global timing metrics, i.e., endpoint arrival time, with the impact of timing optimization taken into account.*

III. Flow Overview

The considerable impact of timing optimization arouses the demand for restructure-tolerant timing estimation. As previously stated, the local supervision information is inconsistent with global timing metrics considering feature mismatching. Consequently, the prior local-view fashion may not fit the real-world scenario where timing optimization is taken into account. Based on the observation that timing endpoints are never replaced during timing optimization and the previous finding [7] that it is easier to model timing optimization's impact globally than locally, we propose to take a global endpoint-wise view to build the framework. Specifically, we present an endpoint embedding flow



Placed Circuit
Endpoint-wise Netlist Information Extraction
v_n
Endpoint-wise Netlist Embeddings
Endpoint Embeddings
v_l
Endpoint-wise Layout Information Extraction
Endpoint-wise Layout Embeddings
Output
MLP
Endpoint Arrival Time Regression

Fig. 2 Overview of our proposed end-to-end endpoint embedding framework, 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 CNN 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.

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 [6].

The overview of our end-to-end framework is shown in Fig. 2, which follows a multimodal fusion paradigm, i.e., combining information from multiple modalities. For each timing endpoint, we encode its netlist information into the netlist embedding $\boldsymbol{v}_n$ with the help of a customized GNN model. Meanwhile, a CNN model and a novel masking technique are utilized to embed global layout information into the endpoint-wise layout embedding $\boldsymbol{v}_l$. The final embedding for the endpoint is given by a combination operation (e.g., concatenation) of $\boldsymbol{v}_n$ and $\boldsymbol{v}_l$. A multi-layer perceptron (MLP) regression model is then applied to consume the generated endpoint embeddings for predicting global timing metrics. We utilize the Mean Squared Error (MSE) as the loss function to guide training:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y_i - p_i)^2, \tag{2}$$

where y_i and p_i are the ground truth arrival time and predicted arrival time for the i_{th} endpoint, respectively.

The algorithm details are introduced in the following sections.

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 with the help of GNN. To correctly reflect the essential timing condition of an endpoint, the information within its whole fanin cone should be embedded, which we refer to as a receptive field of this endpoint. However, the depth of endpoints' fanin cones can range from 2 to more than 400, leading to a considerable variation in the receptive field size from less than 10 pins to thousands of pins. The remarkable variability makes learning endpoint-wise netlist embedding a difficult task worth exploring.

A. Data Representation

Our data transformation follows the prior work [4] that treats each pin as a node since the pin is the essential element in timing analysis. Based on the fact that there are two types of timing arcs, we model the netlist as a heterogeneous graph with two edge types: the net edge and the cell edge. Each net edge represents the connection between a net's drive pin and one of its sink pins. On the other hand, each cell edge connects one of a cell's input pins and its output pin. 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 of sequential elements (e.g., register), there are no cycles in the graph. Consequently, the constructed graph can also 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 to 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: e.g., AND, XOR, embedded as a one-hot vector. (4) Pin capacitance: extracted from the timing library. To facilitate message aggregation, we assign the net distance as the net feature to the net nodes. Similarly, we attach the other three cell features to the cell nodes.

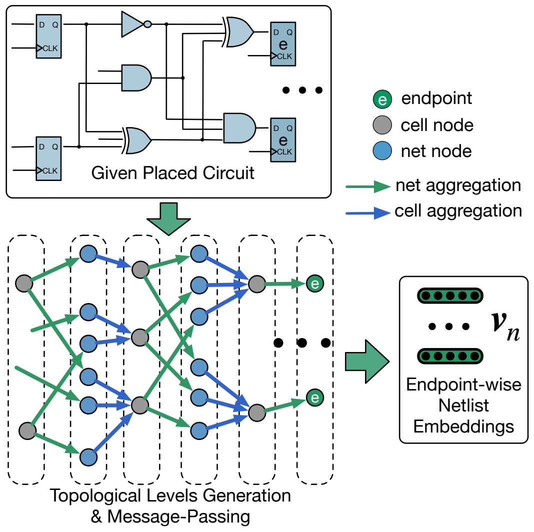


Fig. 3 The message passing scheme of our customized GNN, where the dotted boxes show the graph's topological levels. Two distinct aggregation functions are designed for net edges and cell edges, respectively. The information flows in the topological order, aggregated at the endpoints to generate endpoint-wise netlist embeddings.

B. Customized Graph Neural Network

GNN techniques [17] have provided a paradigm for mining graph information. Since netlists can be represented as graphs, applying GNN for generating endpoint-wise netlist embedding is natural. Nevertheless, it still matters to make customization to correctly reflect the timing condition. Motivated by the delay propagation procedure, our GNN model propagates information in topological order, eventually gathering at the timing endpoints.

Fig. 3 illustrates how to generate netlist embedding for each timing endpoint with our designed GNN. The topological levels of the input graph are first obtained before running the model. Then the message-passing procedure starts from the Primary Inputs (PIs), propagating level by level until all endpoints have been reached. To discriminate between the two timing arcs, we design two distinct message aggregators $\mathcal{A}_c$ and $\mathcal{A}_n$ (Equation (3)) for cell nodes and net nodes, respectively. For cell nodes, we consider a multiple-to-one relationship since each cell node might be driven by multiple input pins. The embedding of a cell node v is composed of two parts: messages from predecessors and information about the corresponding cell (cell features). Based on the fact that the delay at the output pin is only related to the maximum delay of the input pins, we apply a maximum operator to gather messages from the predecessors. On the other hand, the message aggregating scheme along net edges represents a one-to-one relationship since each net node has only one drive node. Similarly, the embedding of a net node v consists of two parts: the received message from the single drive node and the intra-net information (net features). Since the cell node and net node alternate in the propagation flow, the above two message aggregating schemes will take place alternatively.

Formally, the overall message aggregating scheme of a node v can be stated as follows:

$$\boldsymbol{h}_v = \begin{cases} \sigma(\ f_{c1}^{\text{MLP}}(\max\{\boldsymbol{h}_u : u \in \mathcal{N}(v)\}) + f_{c2}^{\text{MLP}}(\boldsymbol{h}_v^c)\ ) & v \in V_c \\ \sigma(\ \boldsymbol{h}_d + f_n^{\text{MLP}}(\boldsymbol{h}_v^n)\ ) & v \in V_n \end{cases} \tag{3}$$

where $\boldsymbol{h}_v$ represents the message (embedding) of node v, V_c is the set of cell nodes, and V_n denotes the set of net nodes. $\boldsymbol{h}_v^c/\boldsymbol{h}_v^n$ is the

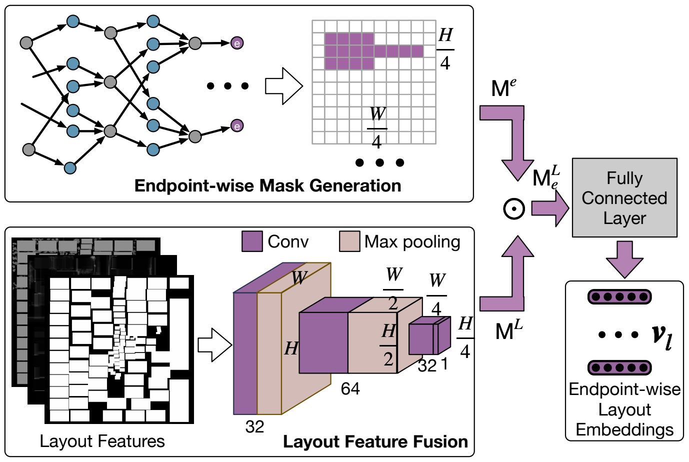


Fig. 4 Our endpoint-wise layout embedding generation flow with a CNN model and a novel endpoint-wise masking technique.

cell/net features of node v, and $\mathcal{N}(v)$ and d denote the predecessor(s) for node v. σ is an activation function, e.g., ReLU. f_{c1}^{MLP}, f_{c2}^{MLP} and f_{n}^{MLP} are MLPs.

V. Layout Information Extraction

Netlist information alone is far from competent to model the complicated timing condition with timing optimization. To boost the model performance, we embed additional information from the layout to capture the impact of timing optimization, as shown in Fig. 4.

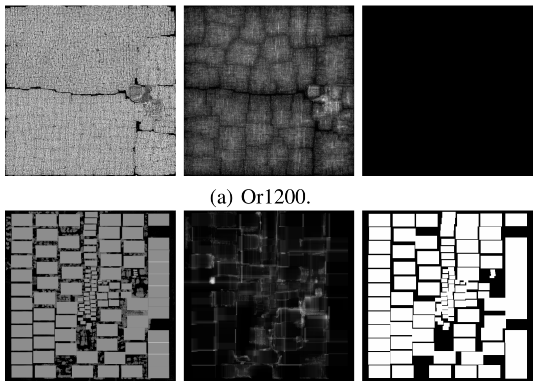
A. *Optimization-related Layout Features*

In general, the selected features are supposed to be closely correlated with timing optimization. Our input features are chosen based on the following observations: 1) most timing optimization techniques need space to insert additional gates, and a denser layout tends to result in a lower timing optimization impact; 2) the macro cell regions cannot be used for timing optimization. As a result, three different features are selected: cell density, rectangular uniform wire density (RUDY), and macro cells region. 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). All three feature maps are derived directly from placed chip layout. Then we stack and feed them to our CNN model (as shown in Fig. 4), yielding fused global layout information map $\mathbf{M}^L \in \mathbb{R}^{\frac{\mathrm{M}}{4}\times\frac{\mathrm{N}}{4}}$ through several convolution and pooling operations.

To illustrate how these three feature maps vary across different circuit designs, Fig. 5 lists 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. It is clear that the layout information for different designs is distinguished.

B. *Endpoint-wise Masking*

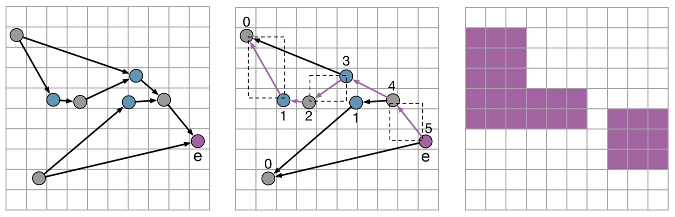
The generated global layout information map $\mathbf{M}^L$ reflects the overall impact of timing optimization. However, it is identical for all the endpoints. In other words, all the endpoints share the same layout information. This does not make sense because the impact of timing optimization varies greatly for different endpoints. For instance, the optimizer tends to prioritize critical endpoints with negative slack, resulting in a more significant impact on these endpoints than on non-critical ones.



(a) Or1200.

(b) Rocket.

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



(a) Input graph (b) Path-finding and mask generation (c) Output mask

Fig. 6 An 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 net edge bounding boxes along P_e.

Based on the above observation, we propose a critical region-based method to extract unique endpoint-wise layout information. Specifically, the layout embedding of an endpoint e only aggregates information from a small region most relevant to e, which is defined as the critical region of e. Motivated by the fact that arrival time at e is closely related to the longest path from PIs to e, we derive the critical region from this longest path.

Fig. 6 gives an example of our proposed endpoint-wise masking technique, which consists of two steps: path-finding and mask generation. While path-finding in undirected graphs is expensive, it is considerably easier in DAGs, especially given that we have already obtained the topological levels (Section IV). Our path-finding algorithm begins by mapping each node to its corresponding topological level. Then we reverse the graph and conduct a depth-first-search (DFS) starting from each endpoint e to find its longest path. During DFS, assume we are visiting node v^i with topological level i, then we move to the successor node p_v^{i-1} with topological level $i-1$ in the next step. If there exist multiple candidates, one of them is randomly picked. Note that a node's topological level implies the distance from PIs to it, hence the chosen successor node p_v^{i-1} is on (one of) the longest path(s) from PIs to v^i. Finally, DFS stops when reaching a PI node, and the visited nodes form the longest path from PIs to endpoint e. Since the above procedure can be conducted in parallel for different endpoints, it is super fast.

It is worth noting that most commonly-used timing optimization techniques are only concerned with layout information outside the

TABLE I Statistics of the dataset, where the left columns depict information about 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, respectively.

Benchmark		Input information				Impact of timing optimization on sign-off metrics					
						slack variation		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%

cells. Therefore, we only consider the region covered by the net edges. After obtaining the longest path P_e for each endpoint e, we construct e's critical region $\mathbf{R}_e$ by taking the union region covered by the bounding boxes of the two-pin net edges along P_e. To be specific, the bounding box for a net edge $\{d, s\}$ is defined as the following rectangular region:

$$\mathbf{B}_{d,s} = [\min\{x_d, x_s\}, \max\{x_d, x_s\}] \times [\min\{y_d, y_s\}, \max\{y_d, y_s\}], \tag{4}$$

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

$$\mathbf{R}_e = \bigcup_{\{d,s\}\in E^n(P_e)} \mathbf{B}_{d,s}, \tag{5}$$

where P_e is e's longest path and $E^n(P_e)$ denotes the net edges along P_e. The critical mask $\mathbf{M}^e \in \mathbb{R}^{\frac{M}{4}\times\frac{N}{4}}$ is then constructed based on R_e as $M^e_{ij} = 1$, $\forall(i,j) \in \mathbf{R}_e$. With the region mask $\mathbf{M}^e$ and the layout information map $\mathbf{M}^L$, the layout map $\mathbf{M}^L_e$ for endpoint e can be calculated as follows,

$$\mathbf{M}^L_e = \mathbf{M}^e \odot \mathbf{M}^L, \tag{6}$$

where $\odot$ denotes Hadamard Product. Then a shared fully connected layer (FCN) is used to convert the layout information map $\mathbf{M}^L_e$ to a low-dimensional layout embedding for e, as shown in Fig. 4.

VI. Experiments

A. Experimental Setup

We develop the framework with DGL [18] and PyTorch [19]. The neural networks are trained and evaluated on a Linux machine with 16 Intel Xeon Gold 6226R cores (2.90GHz), 1 GeForce RTX 3090 Ti graphics card, and 24 GB of main memory. Regarding the hyper-parameters of our model in the experiments, we choose three layers MLPs with a hidden layer dimension of 256 for the GNN part. As for the CNN part, its input size is $3 \times 512 \times 512$, and the network architecture is illustrated in Fig. 4. Besides, the dimension of both endpoint-wise netlist and layout embedding is set to 128, and the MLP used for regression is with 3 layers and a hidden dimension of 512. Our model is trained with a learning rate of 0.001 and batch size of 1024 for 200 epochs.

A dataset of 10 open-source designs is prepared, as listed in TABLE I. We obtain the designs from Chipyard and Github open-source projects. Almost all the designs are RISC-V supported except chacha. We randomly divide the dataset into 5 cases for training and 5 cases for testing. As for the dataset generation flow, we apply Cadence Genus with the edge-cut 7-nm ASAP7 PDK [20] for synthesis, and Cadence Innovus for placement, timing optimization, and routing.

B. Overall Performance

We first compare our method with previous state-of-the-art (SOTA) works [2]–[4] for pre-routing timing evaluation, which are driven by local net/cell delay prediction. As previously stated, netlist restructuring during timing optimization brings troubles to these works. Since there is no way to label the restructured sub-netlists, we adapt the baseline models to our task in a semi-supervised fashion that guides the training with unchanged cells/nets/pins. The previous two-stage methods [2], [3] are supervised by local net/cell delay, prior end-to-end method [4] is supervised by net/cell delay, pin slew, and pin arrival time, and our model is supervised by endpoint arrival time. All the baseline models are well-trained on our training dataset using the official settings in their papers. A layout-only version of our model is implemented by removing the GNN part; similarly, a netlist-only version is implemented by omitting the CNN part of our framework. R^2 score is used for evaluation, which is the most commonly used metric for regression problems. The closer the R^2 score to 1, the better the model performs.

We summarize our findings from the results in TABLE II as follows:

- Our proposed framework vastly outperforms all the baseline approaches on all the benchmarks for sign-off global timing metrics prediction, achieving a performance gain of 25.2%-37.6% on average.
- Previous methods' performance on local net/cell delay prediction is relatively low, demonstrating that it is hard to model the influence of timing optimization locally with only pre-routing information.
- 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.
- Previous methods tend to perform badly when the netlist structure is greatly modified by timing optimization, e.g., chacha.
- Our proposed global-view fashion, i.e., supervised only by global timing information, outperforms prior local-view fashion when timing optimization is taken into account.
- From the ablation study, we find that layout information alone is useless. But together with netlist information, they can suc-

TABLE II Overall 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 [2], [3] incorporate driver cell delay and net delay while [4] 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 [2]	DAC22-he [3]	DAC22-guo [4]	DAC19 [2]	DAC22-he [3]	DAC22-guo [4]	our CNN-only	our GNN-only	our full
arm9	0.0101	-0.5187	-0.2960 / -1.8234	0.6655	0.7304	0.8279	-0.0011	0.8405	**0.8852**
chacha	-0.1389	-0.1008	-0.0813 / -0.2737	0.4406	0.6146	-0.0253	-0.1152	0.7346	**0.9027**
hwacha	0.0519	-0.0323	-0.8003 / -0.8630	0.2752	0.5186	0.7090	-0.0173	0.8022	**0.8623**
or1200	-0.0395	-0.3051	-3.5679 / -0.0924	0.3226	0.4484	0.6776	-0.0019	0.7381	**0.8081**
sha3	0.3941	0.5554	-0.3713 / 0.1230	0.7784	0.7917	0.8464	-0.0058	0.8635	**0.9035**
avg	0.0555	-0.0803	-1.0234 / -0.5859	0.4965	0.6207	0.6071	-0.0283	0.7958	**0.8724**

TABLE III 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 $4000\times$ speedup on average.

design	commercial (20 threads)				ours			
	opt	route	sta	total	pre	infer	total	speedup
jpeg	7863	624922	227	633012	20.63	5.56	26.19	**24170**×
rocket	16239	19161	167	35567	18.53	2.02	20.55	**1731**×
smallboom	9051	53942	152	63145	19.72	4.81	24.53	**2574**×
steelcore	1294	747	20	2061	0.39	1.12	1.51	**1365**×
xgate	338	630	17	985	0.34	0.48	0.82	**1201**×
arm9	305	1825	16	2146	0.88	1.78	2.66	**807**×
chacha	1621	1794	23	3438	0.82	1.20	2.02	**1702**×
hwacha	43883	136946	241	181070	23.89	5.77	29.66	**6105**×
or1200	28641	40291	339	69271	112.20	6.52	118.72	**583**×
sha3	18785	16870	185	35840	24.95	2.58	27.53	**1302**×
avg.	12802	89713	139	102654	22.23	3.184	25.42	**4154**×

cessfully model the impact of timing optimization. This result highlights the efficacy of multimodal fusion in restructure-tolerant timing prediction.

C. 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, machine learning models have overwhelming superiority in efficiency. The results are listed in TABLE III, 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 4154 times faster on average than the commercial tool to evaluate the sign-off global timing metrics at a small cost of around 13% R^2 score loss. Specifically, we can achieve a $24170\times$ speedup on the complex case jpeg.

VII. Conclusion

Prior studies have noted the importance of fast and accurate pre-routing 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 paper has presented a novel endpoint embedding framework with multimodal fusion by utilizing both GNN and CNN to extract netlist and layout information. An end-to-end flow is further developed for pre-routing prediction on sign-off 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. Finally, we should keep a close eye on multimodal fusion in the VLSI design flow for more thorough information mining.

References

[1] J. Rubinstein, P. Penfield, and M. Horowitz, "Signal delay in rc tree networks," *IEEE TCAD*, 1983.
[2] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *Proc. DAC*, 2019.
[3] X. He, Z. Fu, Y. Wang, C. Chang Liu, and Y. Guo, "Accurate timing prediction at placement stage with look-ahead rc network," in *Proc. DAC*, 2022.
[4] 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. DAC*, 2022.
[5] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single pert-like traversal," in *Proc. ICCAD*, 2003.
[6] 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. ISPD*, 2015.
[7] W. Swartz and C. Sechen, "Timing driven placement for large standard cell circuits," in *Proc. DAC*, 1995.
[8] A. Srivastava, D. Sylvester, and D. Blaauw, *Statistical analysis and optimization for VLSI: Timing and power*. Springer, 2005, vol. 59.
[9] D. Sinha, N. V. Shenoy, and H. Zhou, "Statistical gate sizing for timing yield optimization," in *Proc. ICCAD*, 2005.
[10] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI physical design: from graph partitioning to timing closure*. Springer Science & Business Media, 2011.
[11] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi *et al.*, "A graph placement methodology for fast chip design," *Nature*, 2021.
[12] Y. Ma, H. Ren, B. Khailany, H. Sikka, L. Luo, K. Natarajan, and B. Yu, "High performance graph convolutional networks with applications in testability analysis," in *Proc. DAC*, 2019.
[13] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *Proc. DAC*, 2022.
[14] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. CVPR*, 2012.
[15] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. CVPR*, 2015.
[16] J. Fu, H. Zheng, and T. Mei, "Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition," in *Proc. CVPR*, 2017.
[17] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
[18] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," *arXiv preprint arXiv:1909.01315*, 2019.
[19] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.
[20] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, 2016.