2023 Design, Automation & Test in Europe Conference (DATE 2023)

Fast and Accurate Wire Timing Estimation Based on Graph Learning

Yuyang Ye[1], Tinghuan Chen[2,3], Yifei Gao[1], Hao Yan[1], Bei Yu[2], Longxing Shi[1]
[1]Southeast University [2]CUHK [3]CUHK-Shenzhen

***Abstract*—Accurate wire timing estimation has become a bottleneck in timing optimization since it needs a long turn-around time using a sign-off timer. The gate timing can be calculated accurately using lookup tables in cell libraries. In comparison, the accuracy and efficiency of wire timing calculation for complex RC nets are extremely hard to trade-off. The limited number of wire paths opens a door for the graph learning method in wire timing estimation. In this work, we present a fast and accurate wire timing estimator based on a novel graph learning architecture, namely GNNTrans. It can generate wire path representations by aggregating local structure information and global relationships of whole RC nets, which cannot be collected with traditional graph learning work efficiently. Experimental results on both tree-like and non-tree nets demonstrate improved accuracy, with the max error of wire delay being lower than 5 ps. In addition, our estimator can predict the timing of over 200K nets in less than 100 secs. The fast and accurate work can be integrated into incremental timing optimization for routed designs.**

I. Introduction

During the IC design flow, static timing analysis (STA) is crucial for timing closure. However, when a design gets closer to the tape-out stage, more accurate timing analysis is necessary to achieve timing optimization without overdesign for routed design. Path delay calculation in STA is composed of gate and wire timing estimations. Gate timing can be calculated accurately and quickly through interpolating look-up tables in cell libraries. In contrast, wire timing calculation faces the loss of accuracy and efficiency problems, especially on complex RC nets.

Wire timing estimation relies on the global RC net structure formed by parasitic resistances and capacitances [1], [2]. Based on a complex timing model, several commercial STA tools, such as PrimeTime [3] and Tempus [4], are adopted to perform timing analysis path by path in RC nets. Due to the coupling effect, advanced technology nodes bring complicated non-tree net structures with nontrivial loops and an increasing number of parasitic resistances and capacitances. Thus, complex timing models cannot provide accurate wire timing analysis efficiently for a large-scale design on an advanced technology node [5].

To improve analysis efficiency, early work proposes a machine learning-based wire timing estimator [5]. For tree-like nets, the proposed XGBoost model is used to estimate wire timing by taking manually selected RC net structure features as input. However, the proposed loop-breaking algorithm fails to extract important structural information from non-tree nets.

Recently, graph learning has been proposed to fast and accurately perform machine learning tasks by aggregating information from neighborhoods on graph-like data [6]. Moreover, graph learning is used to solve electronic design automation (EDA) problems since the circuit netlist is naturally represented as a graph then structural features are learned and extracted by the graph learning model [7], [8]. In [9]–[11], graph neural networks (GNNs) are developed to verify circuit testability, reliability and manufacturability. Specific blocks, such as arithmetic blocks, are identified by customized GNNs at the netlist level [12], [13]. In addition, GNNs are designed to perform timing analysis [14]–[16]. However, traditional graph learning methods have extremely low efficiency and memory issues to perform inference on all paths in circuits one by one since path number exponentially increases with gate number in a circuit. Besides, many layers need to be stacked to extract global information and perform embedding for long paths. The layer number is at least the maximum node distance within a graph. Such a deep model inevitably brings an over-smoothing issue, which significantly degrades estimation accuracy [17].

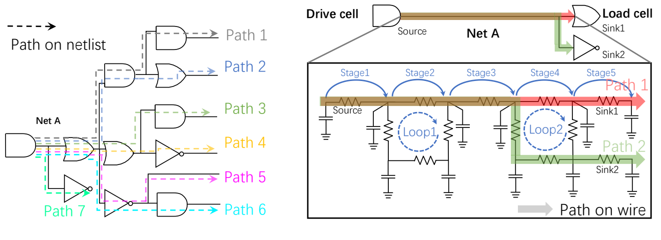

Fig. 1 Examples of (a) All paths on the netlist; (b) All Paths on the wire.

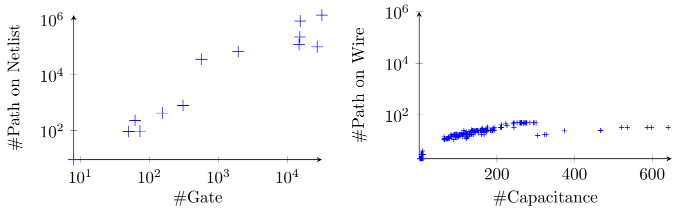

Fig. 2 #Path v.s. (a) #Gate on netlist; (b) #Cap. on wire.

Unlike other EDA problems, there are few paths in each wire RC net. As shown in Fig. 1(a), there are 7 paths on the netlist with 11 gates. In contrast, there are just 2 paths on the wire RC net with 11 capacitances, as shown in Fig. 1(b). From the graph view, gates in a netlist and capacitances in an RC net can be regarded as nodes in a graph, and the path can be defined as a sub-graph consisting of part of nodes. The path number on RC nets is much smaller than that on netlists for large-scale designs, as shown in Fig. 2. According to our statistics on ISCAS89 benchmarks, the numbers of paths are more than 1 million with just 10k gates, as shown in Fig. 2(a). In the open-source circuit with 200k nets, the maximum path number of paths on these nets is just 49, and most of the nets are composed of 10-30 paths, as shown in Fig. 2(b). The limited number of wire paths opens a door for the graph learning method to estimate wire timing effectively while considering path information.

In this paper, we develop a fast and accurate wire timing estimator based on a new graph learning architecture, namely GNNTrans. Compared with other graph-learning-based EDA works [7]–[16], which just collect the information of nodes and edges in neighborhoods and ignore the information of paths, GNNTrans encodes wire paths into path representations through combining path features and aggregated capacitance features in RC nets. Compared with prior wire timing work [5], which sorts RC net structures manually and lacks careful consideration for relationships among elements, GNNTrans learns local net structure information and global relationships among all elements in the whole net using an end-to-end fashion without additional feature engineering. Our method is evaluated on tree-like and non-tree nets of open-source designs when golden timing data is generated through PrimeTime SI mode. We highlight our contributions as follows.

- To the best of our knowledge, this is the first end-to-end graph learning-based model for wire timing estimation, which efficiently exploits the correlations of the wire timing and RC net information.
- We propose GNNTrans to generate representations of wire paths based on local structure information and global relationships in RC nets. Especially, the path features in RC net can be considered directly and sufficiently with a limited computation source which helps improve estimation accuracy and efficiency.
- Our model is evaluated with open-source designs, demonstrating the timing estimation ability to generalize across unseen wires.

II. Preliminaries

A. Problem Formulation

We focus on achieving fast and accurate wire timing estimation in routed design for complex RC net structures (especially with many loops) without using a sign-off timer. To clearly define our problem, we first provide some important definitions as follows.

Definition 1 (Wire Path)**.** *The timing path of a wire, which is from the source to the target sink.*

Definition 2 (Wire Slew)**.** *The time required for a signal of high-to-low or low-to-high transition on a wire is captured from the signal waveform and defined as fall/rise slew.*

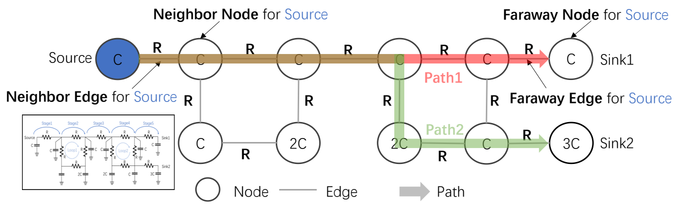

Fig. 3 Equivalent RC graph of Net A shown in Fig. 1(b) with 11 nodes, 12 edges and 2 paths.

Definition 3 (Wire Delay)**.** *The time required for a signal that propagates from the wire source to the target wire sink.*

Now our problem formulation is defined as follows.

Problem 1 (Wire timing estimation)**.** *Given an RC net with parasitics and net structure, capture the information of each path, each capacitance, each resistance, net structure and their relationships effectively and estimate the wire slew and wire delay of the wire path based on these information.*

B. RC Network from Graph View

As shown in Fig. 3, we model a complex RC net as an RC graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{P})$, where each node v_i in $\mathcal{V}$ represents a capacitance, and each edge e_{ij} in $\mathcal{E}$ represents a resistance connected between node v_i and v_j. An edge e_{ij} and its connected node v_j form a stage. $Source$ is the wire path source, and $Sink1$, $Sink2$ are wire path sinks. Thus, the wire path q in $\mathcal{P}$ from the path source $Source$ to the target sink $Sink1$ consists of all nodes and edges visited, which can be cited as a sub-graph. For non-tree nets, the wire path q is the shortest path from the source to the target sink. While other nodes and edges are on the branches.

III. Wire Timing Estimation

A. Overall Flow

In order to handle **Problem 1**, we propose a graph learning method GNNTrans, as illustrated in Fig. 4. It mainly consists of three modules: standard GNN, graph transformer and pooling. The standard GNN extracts local structural information by aggregating features through edges. Then graph transformer extracts global structural information by aggregating all capacitance and resistance in the RC net. Based on the local and global information, the pooling module efficiently generates wire path representation for each path. Multilayer Perceptron layers (MLPs) take the generated representations as input to fast and accurately estimate wire slew and wire delay. Moreover, the circuit timing path arrival time can be estimated by the cumulative addition of our estimated wire delay and cell delay from the timing library.

B. Data Representation

The RC net graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{P})$ is represented with node feature matrix $\boldsymbol{X}$: $\{\boldsymbol{x}_i, \forall i \in \mathcal{V}\}$ for each capacitance, path feature matrix $\boldsymbol{H}$: $\{\boldsymbol{h}_q, \forall q \in \mathcal{P}\}$ for each path and weighted adjacency matrix $\boldsymbol{A} = [a_{i,j}]$. Each element a_{ij} is the value of resistance between node v_i and v_j.

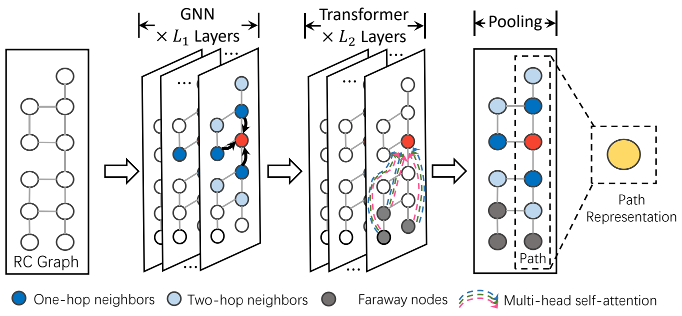

Fig. 4 The architecture of GNNTrans. A GNN module consisting of L_1 layers learns local, short-range structures, then a Graph Transformer module consisting of L_2 layers learns global, long-range relationships. Finally, a pooling module embeds these information and path features to generate wire path representations.

Original Node and Path Features: In order to use GNNTrans, we define an initial node feature vector $\boldsymbol{x}_i \in \mathbb{R}^{d_x \times 1}$ for each capacitance and an initial path feature vector $\boldsymbol{h}_q \in \mathbb{R}^{d_h \times 1}$ for each wire path as listed in TABLE I. d_x and d_h are the dimensions of the node feature vector and path feature vector, respectively. In total, most of the features in the node feature matrix $\boldsymbol{X}$ and path feature matrix $\boldsymbol{H}$ are extracted from RC parasitic results generated through StarRC and design constraints. The downstream capacitance values and stage delays on the path are calculated through the Elmore delay calculation [1]. Downstream capacitance is the accumulated capacitance reachable by resistance on the path. Stage delay is the Elmore delay of each stage. These features are totally chosen based on circuit-domain knowledge and parameter-sweeping experiments.

Weighted adjacency matrix: When there is an edge between between node v_i and v_j, the weight a_{ij} is the value of resistance between capacitance v_i and capacitance v_j.

Labels: The real wire slew $\boldsymbol{S}^{\mathrm{r}}_{\text{wire}}$ and delay $\boldsymbol{D}^{\mathrm{r}}_{\text{wire}}$ for each wire path are generated via PrimeTime with SI mode [3].

An example of our data structure is shown in Fig. 5. Since there are few wire paths in an RC net, no memory issue is caused by our graph learning method while considering node, edge and even path features. In other words, as shown in Fig. 2, unlike gate-level paths, wire paths have few numbers, which allows our graph learning method to perform inference more efficiently without much more memory overhead. Each path information contains its node, edge and path features. By aggregating these features, our model can accurately estimate wire timing on each wire path.

C. GNN Module: Learning Local Structure Information

To learn the RC net graph's local structural information, we update a node's representations by aggregating information from its neighbors with graph connectivity. For different structures, the neighbor information is aggregated together with different methods. There are L_1 GNN layers stacked together to increase the receptive field of the GNN module, which helps learn more structural information. Based on the typical GraphSage models [6], we customize the GNN layers. In GraphSage, each element in the adjacency matrix is binary and only indicates whether there is an edge or not, i.e., connectivities. The node features are always aggregated averagely without considering diverse edge information. In our model, however, each element in the adjacency matrix is the resistance value between two neighbor capacitances, which contains the various edge information. To take advantage of resistances information in RC net, we employ the edge weights while aggregating the representations of the neighbor node for node v_i in the ℓ_1-th GNN layer as follows:

$$\boldsymbol{x}_i^{(\ell_1)} = \mathrm{ReLU}(\boldsymbol{W}_1^{(\ell_1)} \boldsymbol{x}_i^{(\ell_1-1)} + \boldsymbol{W}_2^{(\ell_1)} a_{iu} \sum_{u \in \mathcal{N}(v_i)} \boldsymbol{x}_u^{(\ell_1-1)}), \tag{1}$$

where $\boldsymbol{W}_1^{(\ell_1)}$ and $\boldsymbol{W}_2^{(\ell_1)}$ denote the learnable matrices. ReLU is a nonlinear function. $\boldsymbol{x}_i^{(\ell_1-1)}$ is the node representation of v_i generated in the (ℓ_1-1)-th layer. For the first layer, $\boldsymbol{x}_i^{(0)}$ is the original node feature $\boldsymbol{x}_i$ defined in Section III-B. $\boldsymbol{x}_i^{(\ell_1)}$ is the node representation generated in the ℓ_1-th layer. $\mathcal{N}(v_i)$ is the neighbor node set of v_i. The new aggregation method considers edge information to help the GNNs become more powerful in the 1-Weisfeiler-Lehman (1-WL) isomorphism test. After L_1 GNN layers learning, the pre-node representations $\boldsymbol{X}^{(L_1)}$: $\{\boldsymbol{x}_i^{(L_1)}, \forall i \in \mathcal{V}\}$ are generated.

D. Graph Transformer Module: Learning Global Relationships

GNN module aggregates information from neighbor nodes with existing graph connectivity, which helps learn structure

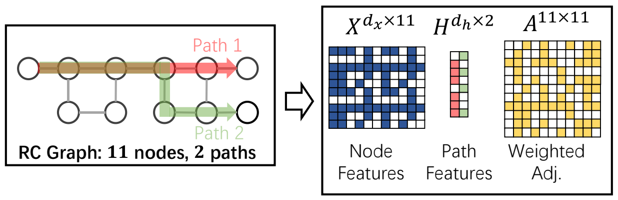

Fig. 5 The example of data representation of RC graph in our work: node feature matrix, path feature matrix and weighted adjacency matrix.

TABLE I Raw node and path features used in GNNTrans.

Type	Name	Description
Node	capacitance value	values of capacitance
	num of input nodes	number of input nodes
	num of output nodes	number of output nodes
	tot input cap	total input capacitance
	tot output cap	total output capacitance
	num of connect. res	number of connected resistance
	tot input res	total input resistance
	tot output res	total output resistance
	downstream cap	Elmore downstream capacitance
	stage delay	Elmore stage delay
Path	input slew	input transition time
	dir. of drive cell	drive strength of drive cell
	func. of drive cell	functionality of drive cell
	dir. of a load cell	drive strength of load cell
	func. of load cell	functionality of load cell
	ceff of load cell	effective capacitance of load cell
	Elmore delay	wire path Elmore delay
	D2M delay	wire path D2M delay

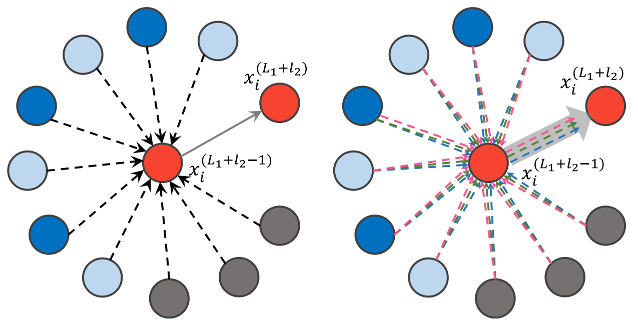

Fig. 6 The illusions of self-attention mechanism (left) and applied multi-head self-attention mechanism (right). In a multi-head self-attention mechanism, the global information is aggregated in two ways: from different nodes and from different representation subspaces (heads).

information. GNNs can aggregate information from beyond local neighborhoods by stacking layers, effectively broadening the GNN receptive field. However, GNN's performance will degrade dramatically when its depth increases, which is an over-smoothing issue [17]. Besides, the global long-range dependencies among each capacitance and resistance cannot be captured by GNNs with a limited receptive field, which will degrade the accuracy of wire timing estimation based on GNNs. Inspired by the self-attention mechanisms' success, we propose to use the multi-head self-attention mechanism in the transformer to learn the global information without an over-smoothing issue.

Once getting the pre-node representations $\boldsymbol{X}^{(L_1)}$: $\{\boldsymbol{x}_i^{(L_1)}, \forall i \in \mathcal{V}\}$, we input them into graph transformer module to learn global relationships among each node. The GNN module aggregates the neighbor information based on the edge weight a_{ij} (the value of resistance between two nodes). Different from the GNN module, the multi-head self-attention mechanism is used to aggregate all element information of the whole RC net and extract global information. The self-attention map $\tilde{\boldsymbol{a}}_{i,u}^{(k,\ell_2)}$ between node v_i and v_j on k-head in $(L_1 + \ell_2)$-th layer is expressed as:

$$\tilde{\boldsymbol{a}}_{i,u}^{(k,\ell_2)} = \operatorname{softmax}\left(\frac{\boldsymbol{W}_Q^{(k,\ell_2)}\boldsymbol{x}_i^{(L_1+\ell_2-1)}(\boldsymbol{W}_K^{(k,\ell_2)}\boldsymbol{x}_u^{(L_1+\ell_2-1)})^T}{\sqrt{d_k}}\right), \tag{2}$$

where $\boldsymbol{W}_Q^{(k,\ell_2)}$, $\boldsymbol{W}_K^{(k,\ell_2)}$ are learnable linear transformation matrices of query and the key for single attention head k in ℓ_2-th transformer layer. $\boldsymbol{X}^{(L_1+\ell_2-1)}$ is the input node representation generated in the last layer and d_k is the dimension of the queries and the keys. For the first layer, the input is the pre-node representations $\boldsymbol{X}^{(L_1)}$. Based on the single self-attention map, the node representation of v_i can be updated based on aggregating information using a multi-head self-attention map:

$$\begin{aligned}\boldsymbol{x}_i^{(L_1+\ell_2)} &= \boldsymbol{x}_i^{(L_1+\ell_2-1)} \\ &+ \boldsymbol{W}_3^{(\ell_2)} \|_{k=1}^{\mathcal{K}} \sum_{u\in\mathcal{V}} \tilde{\boldsymbol{a}}_{iu}^{(k,\ell_2)} \left(\boldsymbol{W}_V^{(k,\ell_2)}\boldsymbol{x}_u^{(L_1+\ell_2-1)}\right),\end{aligned} \tag{3}$$

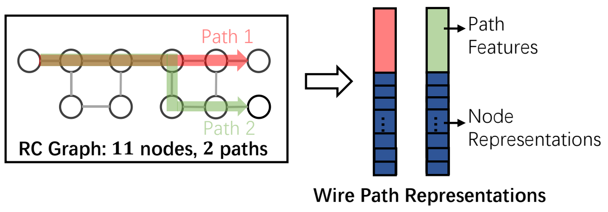

Fig. 7 The illusions of wire representations, which are composed of path features and node representations.

where $\boldsymbol{W}_3^{(\ell_2)}$, $\boldsymbol{W}_V^{(k,\ell_2)}$ are learnable linear transformation matrices. $\|$ denotes concatenating operation. Note that $\boldsymbol{x}_i^{(L_1+\ell_2-1)}$ learns global relationships in transformer layers, because the $u \in \mathcal{V}$ and $\mathcal{V}$ is the node set of given RC net. $\mathcal{K}$ is the number of heads. After L_2 layers, the final node representations $\boldsymbol{X}^{(L_1+L_2)}$: $\{\boldsymbol{x}_i^{(L_1+L_2)}, \forall i \in \mathcal{V}\}$ are generated which contains local structure information and global relationships among all nodes in RC graph. As shown in Equation (3) and Fig. 6, the graph transformer can learn global information in two ways. Firstly, from a graph perspective, the transformer allows every node to attend to every other node, which means it can aggregate information from 'faraway' nodes regardless of the edge connections (totally different from GNN models). More importantly, the graph transformer module can learn the most important node-node relationships globally, instead of favoring nearby nodes. It is different from the GNN module. Secondly, from a single node perspective, multi-head attention ($\mathcal{K}$ attentions in our work) allows the node to jointly attend to information from different representations with different heads.

E. Pooling Module: Representing Wire Paths

In the pooling module, all the node representations $\boldsymbol{X}^{(L_1+L_2)}$: $\{\boldsymbol{x}_i^{(L_1+L_2)}, \forall i \in \mathcal{V}\}$, are selected and combined with original wire path features $\boldsymbol{H}$: $\{\boldsymbol{h}_q, \forall q \in \mathcal{P}\}$ to form wire path representations $\boldsymbol{F}$: $\{\boldsymbol{f}_q, \forall q \in \mathcal{P}\}$. As shown in Fig. 7, the node representations on the wire path are summed up, averaged and concatenated with original path features to generate path representations for each path:

$$\boldsymbol{f}_q = (\frac{1}{N_q}\sum_{v_i\in\mathcal{V}_q}\boldsymbol{x}_i^{(L_1+L_2)})\|\boldsymbol{h}_q, \tag{4}$$

where $\mathcal{V}_q$ is the node set of wire path q and N_q is the number of nodes on wire path q.

F. Predicting Wire Timing

Based on the wire path representations $\boldsymbol{F}$: $\{\boldsymbol{f}_q, \forall q \in \mathcal{P}\}$, we use a multilayer perceptron layer MLP to predict the wire slew and delay under SI mode. For predicting the wire slew for each path, the multilayer perceptron layer takes the wire path representations as input. A trainable parameter $\boldsymbol{\theta}$ in the multilayer perceptron layer MLP is introduced.

$$S_{\mathrm{q}} = MLP(\boldsymbol{\theta} \mid \boldsymbol{f}_q), \tag{5}$$

where S_{q} is the wire slew estimation result of wire path q. For predicting the wire delay for each path, the multilayer perceptron layer takes the wire path representations and the estimated wire slew results as inputs. A new trainable parameter $\boldsymbol{\phi}$ in the multilayer perceptron layer MLP is introduced.

$$D_{\mathrm{q}} = MLP(\boldsymbol{\phi} \mid \boldsymbol{f}_q, S_{\mathrm{q}}), \tag{6}$$

where D_{q} is the wire delay estimation result of wire path q.

IV. Experimental Results

We use PyTorch to implement our models. Our models are trained on a Linux machine with 32 cores and 4 NVIDIA Tesla V100 GPUs in parallel. The total memory used in training is 128GB. Synopsys StarRC extracts RC parasitics, and the golden timing report is generated by Synopsys PrimeTime with SI mode [3] with TSMC16nm technology. The clock period is set to 1.5ns on a 72-core 2.6GHz Linux machine with 1024 GB memory. We use the R^2 score to evaluate the accuracy of the relative wire slew/delay and path arrival time on the testing benchmarks. The larger R^2 score means higher accuracy. In addition, the maximum absolute error of path arrival time is reported.

We train and evaluate our GNNTrans on open-source designs (including Opencore designs [20]). The nets, including tree-like and non-tree nets, are split into training and testing cases as shown in TABLE II. The training process is to minimize the Mean-Squared Error (MSE) between the estimated slew/delay (S_{wire} and D_{wire}) and the ground truths in PrimeTime timing reports ($S^{\mathrm{r}}_{\mathrm{wire}}$ and $D^{\mathrm{r}}_{\mathrm{wire}}$). The overall training progress is end to end, and the trainable matrices and parameters include: $\boldsymbol{W}_1^{(\ell_1)}$ and $\boldsymbol{W}_2^{(\ell_1)}$ ($\forall \ell_1 \in \{1, ..., L_1\}$) in GNN module; $\boldsymbol{W}_Q^{(k,\ell_2)}$, $\boldsymbol{W}_K^{(k,\ell_2)}$, $\boldsymbol{W}_V^{(k,\ell_2)}$, and $\boldsymbol{W}_3^{(\ell_2)}$ ($\forall k \in \{1, ..., \mathcal{K}\}$, $\forall \ell_2 \in \{1, ..., L_2\}$) in Graph Transformer module; $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ in MLP layers. The model training progress consumes about 19 hours on a single GPU. However, the parallel training method on multiple GPUs achieves a 7.2$\times$ speedup on our servers. Our entire prediction procedure is only based on the net information and learned parameters. The inductive model can be shared across different designs without loss of accuracy even if they are unseen.

A. Accuracy of Estimated Wire Slew and Delay

We compare our GNNTrans with prior work [5] and the state-of-the-art graph learning methods, such as GCNII [17], GrapgSage [6], GAT [18] and graph transformer [19], in the wire slew and delay estimation. Note that the residual connections and identify matrix [17] are adopted to alleviate the over-smoothing issue. Open-source cases in TABLE II are used for training and testing. In GNNTrans, we use $L_1 = 20$ GNN layers and $L_2 = 10$ Transformer layers. All other graph learning models are used to generate node representations with search depth $L = 20$ layers (based on the model's performance results). Mean pooling modules are used to generate wire path representations. MLP modules are used to predict wire slew/delay based on wire path representations. The experimental results in TABLE III show that the proposed GNNTrans significantly outperforms all baselines.

TABLE III shows all the wire slew/delay prediction accuracy (R^2 score) results on non-tree nets. Compared with graph learning methods, the accuracy of the traditional machine learning method [5] is extremely low. The main reason is that the loop-breaking algorithm brings much more induced error. Our GNNTrans is significantly better than other graph learning methods. The average R^2 scores of GNNTrans reach 0.978 and 0.970, which outperforms GCNII by 0.148/0.168, GraphSage by 0.112/0.120, and GAT by 0.133/0.150. The primary reason is that they consider only node features and local structure information and has a performance degradation for large-scale designs with long-range elements. Moreover, compared with Graph Transformer, our method achieves gains of 0.165/0.180 on average. We also compare the accuracy of wire timing estimation on all nets, including tree-like and non-tree nets, and the results are shown in TABLE IV. Our method can achieve 0.990 and 0.986 accuracy on average in wire slew and delay estimation. Furthermore, considering complex global relationships without over-smoothing issues, our method significantly improves accuracy on large-scale designs, compared with other GNN methods.

B. Accuracy of Path Arrival Time

The circuit timing path arrival time can be obtained by combining wire delay/slew estimated by our method and gate delay/slew predicted using PrimeTime [3]. TABLE V shows the path arrival time prediction accuracies based on our method and the conventional machine learning method [5]. Note that the baseline results are from the golden timing analysis tool, PrimeTime SI mode [3]. In TABLE V, there is an obvious difference between Primetime timing reports and timing results with the machine learning method [5]. The traditional method [5] cannot achieve an accurate estimation. The average R^2 score is 0.648 and the maximum absolute error reaches 74.59ps

We test GNNTrans with three different configurations: PlanA (L_1=25, L_2=5), PlanB (L_1=20, L_2=10), PlanC (L_1=15, L_2=15). According to the R^2 scores, the accuracy of our work reaches 0.968, 0.985, and 0.981 on average under three different configurations, respectively. Our GNNTrans with more GNN layers can get higher accuracy for small designs. Our GNNTrans with more Transformer layers can get higher accuracy for large-scale designs. And the average maximum absolute errors using different plans are just 3.48ps, 1.93ps and 1.70ps. It shows that our graph learning-based method obtains more accurate path delay results than prior work [5]. And our method is flexible and general to improve estimation performance while solving different designs.

C. Runtime

More importantly, the wire timing estimator costs 55.7s on average for different designs scaling from 40k to 200k nets, as shown in TABLE V. The runtime of our method is just 97.6s while solving 200k nets in the largest design. Compared with predicting gate timing using Primetime SI mode, the runtime of analyzing wire timing is reduced significantly by our method. The fast and accurate method is beneficial to improving efficiency in the physical design optimization flow.

TABLE II Benchmark statistics.

	Benchmark	#Cells	#Nets (Non-tree)	#FFs	#CPs
Train	PCI_BRIDGE	1234	1598 (279)	310	456
	DMA	10215	10898 (1963)	1956	1475
	B19	33785	34399 (8906)	3420	5093
	SALSA	52895	57737 (16802)	7836	9648
	RocketCore	90859	93812 (38919)	16784	12475
	VGA_LCD	56194	56279 (20527)	17054	8761
	ECG	84127	85058 (31067)	14,018	13189
	TATE	184601	185379 (51037)	31,409	27931
	JPEG	219064	231934 (73915)	37,642	36489
	NETCARD	316137	317974 (76924)	87,317	46713
	LEON3MP	341000	341263 (81687)	108,724	50716
	Total	1390111	1075068 (402026)	326470	212766
Test	WB_DMA	40962	40664 (9493)	718	9619
	LDPC	39377	42018 (10257)	2048	7613
	DES_PERT	48289	48523 (9534)	2983	10976
	AES-128	113168	90905 (42657)	10686	24973
	TV_CORE	207414	189262 (53147)	40681	33706
	NOVA	141990	139224 (36482)	30494	39341
	OPENGFX	219064	231934 (62395)	37,642	47831
	Total	810264	782530 (223965)	125252	221890

TABLE III Estimation accuracy of non-tree nets (R^2 score).

Benchmark	Wire Slew/Delay Estimation Accuracy of Non-tree Nets (R^2 score)					
	DAC20 [5]	GCNII [17]	GraphSage [6]	GAT [18]	Trans. [19]	GNNTrans
WB_DMA	0.721/0.693	0.894/0.846	0.912/0.907	0.907/0.872	0.851/0.804	**0.987/0.979**
LDPC	0.714/0.705	0.871/0.829	0.904/0.893	0.881/0.872	0.817/0.781	**0.991/0.985**
DES_PERT	0.703/0.662	0.906/0.871	0.918/0.872	0.897/0.851	0.824/0.807	**0.984/0.975**
AES-128	0.684/0.651	0.824/0.819	0.846/0.829	0.832/0.824	0.807/0.791	**0.979/0.962**
TV_CORE	0.607/0.594	0.738/0.709	0.819/0.806	0.791/0.748	0.795/0.769	**0.969/0.957**
NOVA	0.664/0.631	0.795/0.781	0.834/0.829	0.819/0.802	0.783/0.774	**0.976/0.971**
OPENGFX	0.568/0.537	0.781/0.759	0.827/0.816	0.792/0.773	0.812/0.803	**0.962/0.959**
Average	0.666/0.639	0.830/0.802	0.866/0.850	0.845/0.820	0.813/0.790	**0.978/0.970**

TABLE IV Estimation accuracy of all nets (R^2 score)

Benchmark	Wire Slew/Delay Estimation Accuracy of All Nets (R^2 score)					
	DAC20 [5]	GCNII [17]	GraphSage [6]	GAT [18]	Trans. [19]	GNNTrans
WB_DMA	0.823/0.791	0.915/0.909	0.944/0.921	0.932/0.916	0.912/0.875	**0.999/0.994**
LDPC	0.815/0.797	0.908/0.863	0.925/0.917	0.913/0.907	0.862/0.859	**0.995/0.991**
DES_PERT	0.837/0.822	0.924/0.913	0.927/0.899	0.902/0.899	0.875/0.861	**0.997/0.990**
AES-128	0.802/0.760	0.879/0.867	0.883/0.872	0.845/0.824	0.867/0.854	**0.987/0.982**
TV_CORE	0.795/0.782	0.821/0.810	0.844/0.837	0.831/0.824	0.889/0.876	**0.989/0.986**
NOVA	0.783/0.710	0.854/0.847	0.872/0.865	0.845/0.831	0.876/0.871	**0.984/0.980**
OPENGFX	0.769/0.729	0.835/0.827	0.864/0.851	0.840/0.829	0.897/0.869	**0.982/0.979**
Average	0.803/0.770	0.877/0.862	0.894/0. 880	0.873/0.861	0.882/0.866	**0.990/0.986**

TABLE V Path arrival time estimation accuracy, including R^2 score / MAE (ps), and runtime (s) comparison. "MAE" represents maximum absolute error. PlanA (L_1=25, L_2=5), PlanB (L_1=20, L_2=10), and PlanC (L_1=15, L_2=15) are GNNTrans with 3 different configurations, which helps test our work in different ways.

Benchmark	Path Delay Estimation Accuracy: R^2 score and MAE(ps)					Runtime(s)			
	PrimeTime	Piror Work	Our Work			STA-SI	Our Work		
	STA-SI	DAC20 [5]	PlanA	PlanB	PlanC	Full	Gate	Wire	Total
WB_DMA	1.0000/0.00	0.746/42.45	**0.999/0.57**	0.997/0.59	0.972/1.52	276.7	136.2	**25.1**	161.3
LDPC	1.0000/0.00	0.722/58.21	**0.998/0.64**	0.996/0.67	0.981/0.83	365.9	200.4	**32.9**	233.3
DES_PERT	1.0000/0.00	0.709/37.32	**0.999/0.43**	0.997/0.71	0.983/1.05	386.3	186.7	**27.4**	214.1
AES-128	1.0000/0.00	0.654/71.27	0.954/5.32	0.984/2.32	**0.990/1.14**	593.7	340.5	**56.7**	397.2
TV_CORE	1.0000/0.00	0.527/127.58	0.928/8.56	0.976/4.27	**0.981/3.94**	614.6	400.6	**60.2**	460.8
NOVA	1.0000/0.00	0.604/84.61	0.967/2.64	0.979/1.25	**0.985/0.91**	1133.8	491.2	**87.3**	578.5
OPENGFX	1.0000/0.00	0.574/100.67	0.931/6.18	0.969/3.68	**0.975/2.54**	1185.4	567.3	**97.6**	664.9
Average	1.0000/0.00	0.648/74.59	0.968/3.48	**0.985/1.93**	0.981/1.70	650.91	331.84	**55.31**	387.16

V. Conclusion

In this work, we apply graph learning techniques to estimate wire timing, which helps speed up timing optimization. A new graph learning architecture, called GNNTrans, is proposed to encode wire paths into path representations containing whole net information. Experimental results on open-source designs show that the wire timing estimator based on GNNTrans is accurate meanwhile fast.

Acknowledgement

This work is supported by the National Natural Science Foundation of China (No. 62274034 and 61904030) and The Research Grants Council of Hong Kong SAR (No. CUHK14209420).

References

[1] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of applied physics*, vol. 19, no. 1, pp. 55–63, 1948.

[2] C. J. Alpert, A. Devgan, and C. Kashyap, "A two moment RC delay metric for performance optimization," in *Proc. ISPD*, 2000, pp. 69–74.

[3] Synopsys, "Primetime user guide," http://www.synopsys.com/Tools/Implementation/SignOff/Documents/primetime_ds.pdf, 2015.

[4] Cadence, "Tempus user guide." https://www.cadence.com/content/tempustiming-signoff-solution.html, 2015.

[5] H.-H. Cheng, I. H.-R. Jiang, and O. Ou, "Fast and accurate wire timing estimation on tree and non-tree net structures," in *Proc. DAC*, 2020.

[6] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Proc. NIPS*, vol. 30, 2017.

[7] T. Chen, G. L. Zhang, B. Yu, B. Li, and U. Schlichtmann, "Machine learning in advanced IC design: A methodological survey," *IEEE MDAT*, 2022.

[8] Y. Ma, Z. He, W. Li, L. Zhang, and B. Yu, "Understanding graphs in EDA: From shallow to deep learning," in *Proc. ISPD*, 2020.

[9] 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.

[10] T. Chen, Q. Sun, C. Zhan, C. Liu, H. Yu, and B. Yu, "Deep H-GCN: Fast analog IC aging-induced degradation estimation," *IEEE TCAD*, 2022.

[11] S. Sun, Y. Jiang, F. Yang, B. Yu, and X. Zeng, "Efficient hotspot detection via graph neural network," in *Proc. DATE*, 2022.

[12] Z. He, Z. Wang, C. Bail, H. Yang, and B. Yu, "Graph learning-based arithmetic block identification," in *Proc. ICCAD*, 2021.

[13] Z. Wang, Z. He, C. Bai, H. Yang, and B. Yu, "Efficient arithmetic block identification with graph learning and network-flow," *IEEE TCAD*, 2022.

[14] K. K.-C. Chang, C.-Y. Chiang, P.-Y. Lee, and I. H.-R. Jiang, "Timing macro modeling with graph neural networks," in *Proc. DAC*, 2022.

[15] 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.

[16] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi, "Graph-learning-driven path-based timing analysis results predictor from graph-based timing analysis," in *Proc. ASPDAC*, 2023.

[17] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proc. ICML*, 2020.

[18] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *stat*, vol. 1050, p. 20, 2017.

[19] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *arXiv preprint arXiv:2012.09699*, 2020.

[20] OpenCores, http:///opencores.org/, 2021.