Graph-Learning-Driven Path-Based Timing Analysis Results Predictor from Graph-Based Timing Analysis

Yuyang Ye[1], Tinghuan Chen[2], Yifei Gao[1], Hao Yan[1], Bei Yu[2], Longxing Shi[1]
[1] School of Electronic Science and Engineering, Southeast University
[2] Department of Computer Science and Engineering, Chinese University of Hong Kong
{yeyuyang, g_yifei, yanhao, lxshi}@seu.edu.cn; {thchen, byu}@cse.cuhk.edu.hk

Abstract

With diminishing margins in advanced technology nodes, the performance of static timing analysis (STA) is a serious concern, including accuracy and runtime. The STA can generally be divided into graph-based analysis (GBA) and path-based analysis (PBA). For GBA, the timing results are always pessimistic, leading to overdesign during design optimization. For PBA, the timing pessimism is reduced via propagating real path-specific slews with the cost of severe runtime overheads relative to GBA. In this work, we present a fast and accurate predictor of post-layout PBA timing results from inexpensive GBA based on deep edge-featured graph attention network, namely deep EdgeGAT. Compared with the conventional machine and graph learning methods, deep EdgeGAT can learn global timing path information. Experimental results demonstrate that our predictor has the potential to substantially predict PBA timing results accurately and reduce timing pessimism of GBA with maximum error reaching 6.81 ps, and our work achieves an average 24.80× speedup faster than PBA using the commercial STA tool.

ACM Reference Format:
Yuyang Ye, Tinghuan Chen, Yifei Gao, Hao Yan, Bei Yu, Longxing Shi. 2023. Graph-Learning-Driven Path-Based Timing Analysis Results Predictor from Graph-Based Timing Analysis . In *28th Asia and South Pacific Design Automation Conference (ASPDAC '23), January 16–19, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3566097.3567904

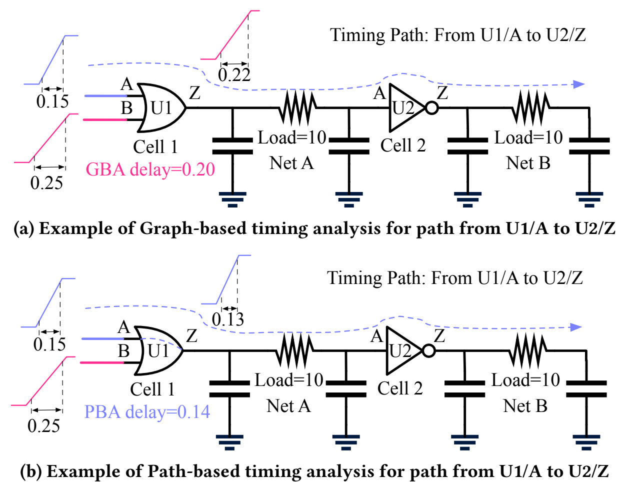

Figure 1: Examples of timing analysis in GBA and PBA modes: The GBA mode is fast, while the results are pessimistic; For PBA mode, timing analysis pessimism is reduced at the cost of runtime increment.

1 Introduction

The accuracy of timing analysis is a serious concern for diminishing margins in advanced technology nodes for designs with over millions of cells and many multi-corner multi-mode timing scenarios. Improved accuracy helps to reduce overdesign, particularly during place-and-route and timing closure steps. However, it comes at the cost of long runtime while timing analysis. Commercial EDA tools, such as PrimeTime [1] and Tempus [2], support graph-based analysis (GBA) and path-based analysis (PBA) modes in static timing analysis (STA), enabling a tradeoff of accuracy versus runtime.

In GBA mode, pessimistic transition time is propagated at each cell of the timing graph [1]. GBA is able to calculate all the path delays in circuits and report the critical paths in a fast way. However, it always introduces pessimism due to the worst-case slew propagation. A GBA example is shown in Figure 1(a), where the worst slew of input pins is utilized in slew propagation. During a setup check for the timing path from U1/A to U2/Z, the slew of pin U1/Z would be computed based on the worst slew, i.e. slew at U1/B, thus the slew of the pin U1/Z will be overestimated. Then the delay of cells on the timing path, including U1 and U2, will be overestimated, which causes path arrival time overestimated, leading to an overdesign.

In PBA mode, timing analysis pessimism is reduced at the cost of significantly greater runtime than GBA mode [3]. Figure 1(b) illustrates transition propagation in PBA mode. For the same timing path, the slew of the pin U1/Z is computed based on the real slew of inputs, i.e. slew at U1/A. Then the actual path-specific slew is propagated on the path and used in delay calculation for cells, which helps to improve the accuracy of time analysis. As the number of timing paths increases, there is an exponential increment in the possibilities of transition propagation and delay calculation at each cell. It causes PBA runtime-intensive.

PBA plays an important role in critical path timing signoff for large-scale designs using advanced technology nodes. Unfortunately, it is difficult to tradeoff accuracy and runtime while timing analysis due to today's timing-consuming PBA. In Molina [4] and Kahng [5], they name fast prediction of PBA timing results based GBA results as a solution to achieve runtime and accuracy tradeoff and important near-term challenge for machine learning applied in electronic design automation (EDA) community. Kahng et al. [6] develop two

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASPDAC '23, January 16–19, 2023, Tokyo, Japan
© 2023 Association for Computing Machinery.
ACM ISBN 978-1-4503-9783-4/23/01…$15.00
https://doi.org/10.1145/3566097.3567904

classification and regression tree models to capture divergence in cell slew/delay in PBA and GBA timing mode, where the overall timing path delay divergence is estimated by cumulative cell delay divergence values on timing paths. However, the accuracy is extremely limited while only considering local cell features.

Recently, graph neural networks (GNNs) are proposed to fast and accurately perform machine learning tasks through aggregating information from neighborhoods on graph-like data [7]. Since circuits can be represented as graphs directly, GNNs are used to solve various electronic design automation (EDA) problems [8–11]. However, there is a common problem in the current GNN models, such as graph attention networks (GATs) and graph convolutional networks (GCNs), is that edge features are not fully considered. In GATs, the edge information considered is only the indication of whether there is an edge or not, i.e., connectivities. In GCNs, the edge features utilized are just one-dimensional values, e.g., edge weights. However, post-layout circuit graph edges often possess rich information, which represents the interconnects as shown in Figure 1. Instead of being a binary variable or one-dimensional value, the edge features should be continuous and multi-dimensional while achieve timing analysis, including the parasitic capacitances and resistances information, which influences cell slew and delay significantly.

In this paper, we propose a fast predictor of post-layout PBA timing results from GBA results based on deep EdgeGAT. Compared with prior work [6] which predicts PBA results based on local cell features, our GAT model can help learn global and complete timing information, including cell features and net features on whole timing paths. Totally different from conventional GNN models, the proposed deep edge-featured graph attention network model, namely deep EdgeGAT, is capable of exploiting multi-dimensional node (cell) and edge (net) features with a new attention mechanism to incorporate them into better node embeddings for predicting PBA timing results in a fast and accurate way. We highlight our contributions in detail.

- To the best of our knowledge, we are the first to present an end-to-end GAT-based model for accurately predicting path-based timing values, including cell slew, delay and path arrival time, from graph-based timing analysis. It helps reduce pessimism in GBA and avoid overdesign in design.
- We present deep edge-featured graph attention networks, namely deep EdgeGAT, to extend traditional GATs and to perform embedding for predicting PBA results accurately with learning both node (cell) and edge (net) features.
- We predict cell slew and delay in a single model synergistically to facilitate model training.
- Our predictor is evaluated with real designs, which demonstrates the generalize ability to solve unseen designs.

2 Problem Formulation and Discussions

2.1 Problem Formulation

Compared with cell delay while calculating path arrival time, the divergence of wire delay under GBA and PBA mode can be ignorable. Thus, we focus on predicting PBA cell slew/delay and path arrival time based on GBA timing reports and timing path structure accurately for critical paths (CPs) to achieve timing analysis efficiency and accuracy tradeoff. For each cell in a design, the problem is to capture complex interactions of the input node/edge features and their impact on PBA timing results. If the predictor is trained using a sufficient number of designs, it can be applied to unseen designs with generalization. Since deep EdgeGAT is used in our predictor for improving accuracy, we represent all critical paths (CPs) in circuits as an edge-featured graph and take it as input of deep EdgeGAT, which we introduce in detail as follows.

Definition 1 (Edge-featured Graph)**.** An edge-featured graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \boldsymbol{X}, \boldsymbol{H}\}$ is defined as an undirected graph consisting of: (1) a node set $\mathcal{V} = \{v^{(1)}, v^{(2)}, \ldots, v^{(n)}\}$, where $|\mathcal{V}| = N$. It denotes cell set on critical paths; (2) an edge set $\mathcal{E}$, where $|\mathcal{E}| = M$. It denotes net set on critical paths; (3) node features $\boldsymbol{X} \in \mathbb{R}^{n \times k_x}$, where i^{th} row vector $\boldsymbol{x_i} \in \mathbb{R}^{k_x}$ is the node features for the ith node; (4) edge features $\boldsymbol{H} \in \mathbb{R}^{m \times k_h}$, where the row vector $\boldsymbol{h_p} \in \mathbb{R}^{k_h}$ is the edge features for the pth edge or the edge between i^{th} and j^{th} node.

In this paper, an edge-featured graph $\mathcal{G}$ is represented by an edge-featured adjacency matrix $\boldsymbol{H}$, a node feature matrix $\boldsymbol{X}$, and a label matrix $\boldsymbol{y}$, including cell slew and delay results. Based on this definition, the problem formulation is shown as follows.

Problem 1. Given a training set P_{train} which includes edge-featured graphs representing critical paths with GBA and PBA timing results in training cases, we use P_{train} to train a graph-learning based model for a testing set P_{test} (where $P_{\text{test}} \cap P_{\text{train}} = \emptyset$) which includes edge-featured graphs representing critical paths with GBA results in testing cases and generate their PBA timing results based on given GBA timing results and timing path structure information without additional STA runtime.

2.2 Overall Flow

In this paper, **Problem 1** is divided into three tasks based on delay calculation progress: Task 1 performs node embedding considering each cell, net information on critical paths and path structure based on trained deep EdgeGAT, which is beneficial to improve predicting accuracy of cell slew and delay in PBA mode based on GBA results. Task 2 predicts cell slew in PBA mode $S^{\text{PBA}}_{\text{cell}}$ for each cell based on node embedding results at first, then uses the predicted cell slew along with node embedding results to predict PBA cell delay $D^{\text{PBA}}_{\text{cell}}$. In Task 3, the path arrival time of target critical path in PBA mode $AT^{\text{PBA}}_{\text{path}}$ can be estimated by cumulative addition of predicted PBA cell delay $D^{\text{PBA}}_{\text{cell}}$ values and GBA wire delay $D^{\text{GBA}}_{\text{wire}}$ values on it. The overall flow is shown in Figure 2.

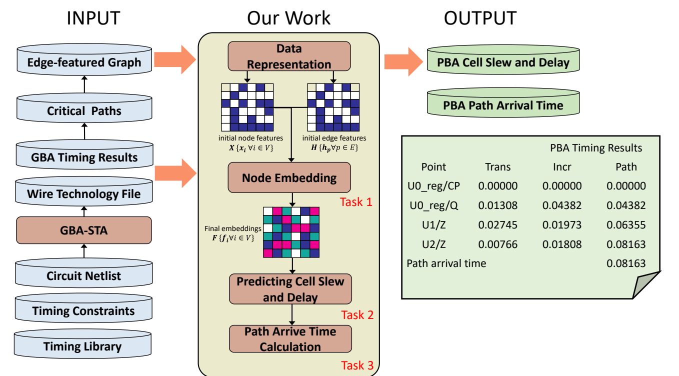

Figure 2: The overall flow of our PBA timing results predictor.

Table 1: Raw node and edge features used in deep EdgeGAT.

Type	Name	Description
Node	cell delay	delay of cell
	cell output slew	transition time of cell output pin
	cell input slew	transition time of cell input pin on path
	cell input slew type	rise or fall
	cell threshold voltage	threshold voltage of cell
	wst cell input slew	worst transition time of input pins
	cell drive strength	drive strength of cell
	cell functionality	functionality of cell
	tot cell input cap	sum of cell input pin cap
	tot cell load cap	total load capacitance of cell
Edge	net delay	delay of net
	net slew type	rise or fall
	net output slew	transition time of net output pin
	net input slew	transition time of net input pin
	tot net cap	sum of net capacitance
	tot net res	sum of net resistance
	net input cap	capacitance of driver cell for net
	tot net load cap	total capacitance of load cells

2.3 Why Deep Graph Learning?

In a D-layer graph learning neural network, each node can learn information in the D-hop neighborhood. Thus information beyond the D-hop neighborhood can not be aggregated. In [6], it uses 2-stage model, namely bigram-based model, as the fundamental unit for predicting PBA timing. From the perspective of graph learning, it can be regarded as a simple $D = 2$ model, where only the features of driving cells and loading cells can be aggregated to the node embedding of the target cell. Thus, the model proposed in [6] cannot encode global path timing features and structure in node embeddings, which causes inaccurate timing prediction results. For collecting timing information on paths, the number of layers should be equal to the maximum timing depth, where the number is about 100 for circuits with millions of cells [9]. More importantly, there are just additional 2 neighbors in each hop on timing path graph and the number of neighbors of a node scales linearly as the hop-count increases, which is totally different from exponential increment of existing EDA works [8–11]. The over-smoothing problem in deep graph learning is solved through residual connections and identity matrix inspired by GCNII [12].

3 PBA Predictor Based on Deep EdgeGAT

Given critical paths represented with an edge-featured graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \boldsymbol{X}, \boldsymbol{H}\}$, we predict PBA timing results from GBA timing results based on node and edge features using deep EdgeGAT and calculate path arrival time for critical paths with ignorable additional runtime. As shown in Figure 2, the prediction process is composed of node embedding, cell slew/delay predicting and path delay calculation. We overcome the issue of the limited consideration of edge features in conventional graph learning models by using node and edge attention aggregator during node embedding. Thus, our predictor has high explainability and strong generalization power since it fully learns path information, including cell features, net features and path structure. Our trained work can predict PBA timing results accurately and reduce pessimism in GBA.

3.1 Data Representation

Original Node Features and Edge Features: Before leveraging deep EdgeGAT to generate better node embedding, we define an initial node feature vector $\boldsymbol{x}_i$ for each cell and an initial edge feature vector $\boldsymbol{h}_p$ for each net as shown in Table 1. In total, the 18 features in Table 1 are extracted from technology files, SPEF files, and PrimeTime GBA timing reports. These features are chosen based on circuit knowledge and parameter-sweeping experiments. However, the predicted PBA timing results are not accurate just based on these manually engineered features. We leverage deep EdgeGAT to perform graph learning to achieve better node embedding.
Labels: The real cell slew $S_{\mathrm{r}}^{\mathrm{PBA}}$ and delay $D_{\mathrm{r}}^{\mathrm{PBA}}$ for each cell on critical paths under PBA mode, which is generated via golden commercial STA tool PrimeTime [1].

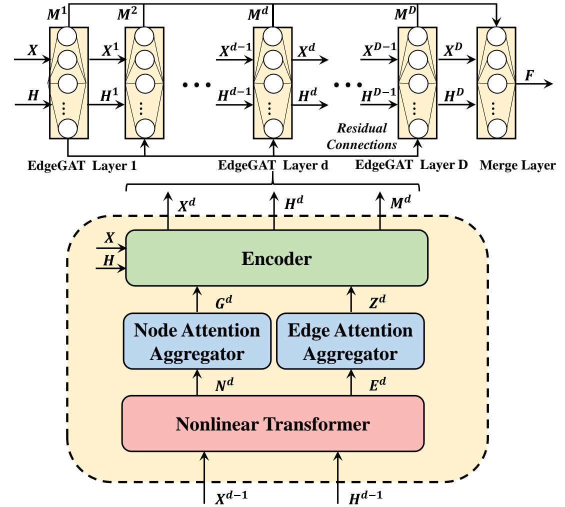

Figure 3: The architecture of node embedding framework, including D EdgeGAT layers and merge layer. A single EdgeGAT layer is composed of nonlinear transformer, node attention aggregator, edge attention aggregator and encoder.

3.2 Node Embedding

To predict the cell slew and delay accurately, **EdgeGAT layers** and **merge layer** in deep EdgeGAT are used to generate new node embedding $\boldsymbol{F}$: $\{\boldsymbol{f}_i, \forall i \in \mathcal{V}\}$ for cells in circuit which is based on node (cell) features $\boldsymbol{X}$: $\{\boldsymbol{x}_i, \forall i \in \mathcal{V}\}$, edge (net) features $\boldsymbol{H}$: $\{\boldsymbol{h}_p, \forall p \in \mathcal{E}\}$, and timing path structural information. Since the node embedding is expected to aggregate the information from multiple hop neighbors for getting more accurate representations. According to the principles of graph neural networks, D EdgeGAT layers are used to aggregate D-hop neighborhoods in our work, as shown in Figure 3. In our work, $D = 100$ according to the number of timing path logic depths. Next, we introduce how the global node and edge information are aggregated and achieve node embedding without over-smoothing in d-th EdgeGAT layer at details.

3.2.1 EdgeGAT Layers. Figure 3 gives an detailed illustration of d-th EdgeGAT layer, it takes node feature matrix $\boldsymbol{X}^{d-1}$: $\{\boldsymbol{x}_i^{d-1}, \forall i \in \mathcal{V}\}$ and edge feature matrix $\boldsymbol{H}^{d-1}$: $\{\boldsymbol{h}_p^{d-1}, \forall p \in \mathcal{E}\}$ generated in (d-1)-th EdgeGAT layer as inputs and outputs $\boldsymbol{X}^d$: $\{\boldsymbol{x}_i^d, \forall i \in \mathcal{V}\}$, $\boldsymbol{H}^d$: $\{\boldsymbol{h}_p^d, \forall p \in \mathcal{E}\}$ and $\boldsymbol{M}^d$: $\{\boldsymbol{m}_i^d, \forall i \in \mathcal{V}\}$. For the first EdgeGAT layer, the input is the original node feature matrix $\boldsymbol{X}$ and edge matrix $\boldsymbol{H}$ defined in Section 3.1. In order to alleviate the over-smoothing

issue in deep EdgeGAT layers, the input of the first layer should be concatenated with each layer as residual connections, which ensures the final representation of each node retains the input layer even if many layers are stacked. A single EdgeGAT layer contains four different modules: nonlinear transformer, node attention aggregator, edge attention aggregator, and encoder. Each EdgeGAT layer is designed in a symmetrical scheme; thus, the node and edge embeddings can update themselves in a parallel and equivalent way. We will introduce four different modules in detail as follows.

Nonlinear Transformer: To achieve nonlinear transforming in the d-th EdgeGAT layer, two learnable matrices, $\boldsymbol{W}_X^d \in \mathbb{R}^{K_X^d \times K_X^{d-1}}$, $\boldsymbol{W}_H^d \in \mathbb{R}^{K_H^d \times K_H^{d-1}}$ and a hyper-parameter l^d, are used to transform the input node features $\{\boldsymbol{x}_i^{d-1} \in \mathbb{R}^{K_X^{d-1}}, \forall i \in \mathcal{V}\}$ and edge features $\{\boldsymbol{h}_p^{d-1} \in \mathbb{R}^{K_H^{d-1}}, \forall p \in \mathcal{E}\}$ into latent representations $\boldsymbol{n}_i^d$ and $\boldsymbol{e}_i^d$:

$$\boldsymbol{n}_i^d = ((1-l^d)\boldsymbol{I} + l^d \boldsymbol{W}_X^d) \cdot \boldsymbol{x}_i^{d-1}, \boldsymbol{e}_p^d = ((1-l^d)\boldsymbol{I} + l^d \boldsymbol{W}_H^d) \cdot \boldsymbol{h}_p^{d-1}, \quad (1)$$

where the identity matrix $\boldsymbol{I}$ is easy to extend as an augmented matrix of an identity matrix with a zero matrix if $\boldsymbol{W}_X^d$ and $\boldsymbol{W}_H^d$ are not square matrices. Compared with linear transformer used in traditional GAT, the matrix $\boldsymbol{I}$ and hyper-parameter l^d help to overcome over-smoothing issue in deep graph learning.

Node Attention Aggregator: The node attention aggregator in d-th EdgeGAT layer accepts the transformed node and edge representations generated through liner transformer as inputs, $\{\boldsymbol{n}_i^d, \forall i \in \mathcal{V}\}$ and $\{\boldsymbol{e}_i^d, \forall p \in \mathcal{E}\}$, and produces aggregated node representations $\{\boldsymbol{g}_i^d, \forall i \in \mathcal{V}\}$. For generating aggregated node representations, we aggregate the node representations from the node's neighbors via node attention coefficients α^d which indicate the importance of neighborhood information to the target node.

$$\alpha_{ij}^d = \frac{\exp\left(\text{LeakyReLU}\left((\boldsymbol{a}^d)^\top \left[\boldsymbol{n}_i^d \| \boldsymbol{n}_j^d \| \boldsymbol{e}_{ij}^d\right]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left((\boldsymbol{a}^d)^\top \left[\boldsymbol{n}_i^d \| \boldsymbol{n}_k^d \| \boldsymbol{e}_{ik}^d\right]\right)\right)}, \quad (2)$$

where i is the target node and the j is its neighbor belongs to neighborhood set N_i. $\cdot^\top$ represents transposition and $\|$ is the concatenation operation. The node attention coefficient is parametrized by a weight vector $\boldsymbol{a}_d \in \mathbb{R}^{2K_X^d + K_H^d}$ with the LeakyReLU nonlinear function (negative input slope equals to 0.2). And all the node attention coefficients of node i are normalized with $softmax$ function.

Based on the normalized coefficients for each neighborhood, we can perform a weighted sum on these node representations of neighbors in $\mathcal{N}_i$ to get aggregated node representation $\boldsymbol{g}_i^d$ for node i:

$$\boldsymbol{g}_i^d = \sum_{j \in \mathcal{N}_i} \alpha_{ij} \boldsymbol{n}_j^d, \quad \forall i \in \mathcal{V}. \quad (3)$$

Edge Attention Aggregator: The node representations are aggregated in the node attention aggregator to acquire better node embeddings, while the original edge features are necessary to be updated during the weight computation. Thus we update edge representations to help learn global edge information while node embedding. We propose an edge attention aggregator in d-th EdgeGAT layer, which accepts the same input as node attention aggregator, including $\{\boldsymbol{n}_i^d, \forall i \in \mathcal{V}\}$ and $\{\boldsymbol{e}_p^d, \forall p \in \mathcal{E}\}$. Different from node attention module, edge-attention module produces aggregated edge representations $\{\boldsymbol{z}_p^d, \forall p \in \mathcal{E}\}$ based on edge attention coefficients β.

$$\beta_{pq}^d = \frac{\exp\left(\text{LeakyReLU}\left((\boldsymbol{b}^d)^\top \left[\boldsymbol{e}_p^d \| \boldsymbol{e}_q^d \| \boldsymbol{n}_{pq}^d\right]\right)\right)}{\sum_{k \in \mathcal{N}_p} \exp\left(\text{LeakyReLU}\left((\boldsymbol{b}^d)^\top \left[\boldsymbol{e}_p^d \| \boldsymbol{e}_k^d \| \boldsymbol{n}_{pk}^d\right]\right)\right)}, \quad (4)$$

where p is the target edge and the q is its neighbor edge which belongs to neighborhood set N_p. The edge attention coefficient parametrized by a weight vector $\boldsymbol{b}_d \in \mathbb{R}^{2K_H^d + K_X^d}$ with the LeakyReLU nonlinear function (negative input slope equals to 0.4). We can get the aggregated representation $\boldsymbol{z}_p^d$ for edge p:

$$\boldsymbol{z}_p^d = \sum_{q \in \mathcal{N}_p} \beta_{pq} \boldsymbol{e}_q^d, \quad \forall p \in \mathcal{E}. \quad (5)$$

Encoder: A non-linear transformation σ is performed to encode the aggregated node and edge representations in this module. In our work, *ReLU* function is used. After encoding, we can get new node feature matrix $\boldsymbol{X}^d$: $\{\boldsymbol{x}_i^d, \forall i \in \mathcal{V}\}$, edge feature matrix $\boldsymbol{H}^d$: $\{\boldsymbol{h}_p^d, \forall p \in \mathcal{E}\}$, and edge-integrated feature matrix $\boldsymbol{M}^d$: $\{\boldsymbol{m}_i^d, \forall i \in \mathcal{V}\}$.

$$\boldsymbol{x}_i^d = \sigma(\boldsymbol{g}_i^d \| \boldsymbol{x}_i), \boldsymbol{h}_p^d = \sigma(\boldsymbol{z}_i^d \| \boldsymbol{h}_i), \boldsymbol{m}_i^d = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \left(\boldsymbol{n}_j \| \boldsymbol{e}_{ij}\right)\right). \quad (6)$$

3.2.2 Merge Layer. After the process of EdgeGAT layers, we can get the final node embedding results $\boldsymbol{F}$: $\{\boldsymbol{f}_i, \forall i \in \mathcal{V}\}$ in merge layer based on each edge-integrated feature matrix $\boldsymbol{M}^d$: $\{\boldsymbol{m}_i^d, \forall i \in \mathcal{V}\}$ generated through each EdgeGAT layer.

$$\boldsymbol{f}_i = \|_{d=1}^D (\boldsymbol{m}_i^d), \quad \forall i \in \mathcal{V}. \quad (7)$$

3.3 Predicting Cell Slew and Delay

Based on the final node embedding results $\boldsymbol{F}$: $\{\boldsymbol{f}_i, \forall i \in \mathcal{V}\}$, we use a multilayer perceptron module (*MLP*) to predict the cell slew and delay in PBA mode. The objective of *MLP* can be expressed as minimizing Mean-Squared Error (MSE) between the predicted slew/delay and the PBA slew/delay. Thus, the loss function of *MLP* can be divided into two parts. For predicting PBA cell slew, the loss function is shown in Equation (8), where $\boldsymbol{F}$ is the node embedding results generated in Section 3.2, N is the number of nodes (cells) in given circuit, $S_{\text{cell}}^{\text{PBA}}$ is the cell slew prediction result of multilayer perceptron layer, $S_{\text{r}}^{\text{PBA}}$ is the real PBA slew of cells in label matrix $\boldsymbol{y}$, $\boldsymbol{\theta}$ is trainable parameter set of for PBA cell slew predicting.

$$\mathcal{L}_{\text{slew}}(\boldsymbol{\theta} \mid \boldsymbol{F}, S_{\text{r}}^{\text{PBA}}) = \frac{1}{N} \sum_{i \in \mathcal{V}} (S_{\text{i}}^{\text{PBA}} - S_{\text{r_i}}^{\text{PBA}})^2. \quad (8)$$

Additionally, cell delay prediction is the other task of the multilayer perceptron module (*MLP*). Equation (9) shows the objective to minimize Mean-Squared Error (MSE) between predicted cell delay and the ground truth, where $D_{\text{cell}}^{\text{PBA}}$ is the cell delay prediction result of multilayer perceptron layer, $D_{\text{r}}^{\text{PBA}}$ is the ground truth PBA delay of cells in label matrix $\boldsymbol{y}$, and $\boldsymbol{\phi}$ is trainable parameter set for PBA cell delay predicting. This step helps the prediction model to learn the cell delay computation progress in PBA mode based on cell slew and other GBA timing results.

$$\mathcal{L}_{\text{delay}}(\boldsymbol{\phi} \mid \{\boldsymbol{F}, S_{\text{cell}}^{\text{PBA}}\}, D_{\text{r}}^{\text{PBA}}) = \frac{1}{N} \sum_{i \in \mathcal{V}} (D_{\text{i}}^{\text{PBA}} - D_{\text{r_i}}^{\text{PBA}})^2. \quad (9)$$

Finally, we can get the overall loss function by combining cell slew and delay predicting, as shown in Equation (10). The two components are minimized simultaneously and help improve prediction accuracy.

$$\mathcal{L}_{\text{tot}}(\boldsymbol{\theta}, \boldsymbol{\phi} \mid \{\boldsymbol{F}, S_{\text{cell}}^{\text{PBA}}\}, S_{\text{r}}^{\text{PBA}}, D_{\text{r}}^{\text{PBA}}) = \mathcal{L}_{\text{slew}} + \mathcal{L}_{\text{delay}}. \tag{10}$$

3.4 Parallel Training and Inference

Algorithm 1 summarizes the overall training process of PBA cell slew/delay predictor using open-source cases. Lines 1-13 illustrate the node embedding process for each node $v \in \mathcal{V}_t$, including nonlinear transformer, node attention aggregator, edge attention aggregator, and encoder. When maximum depth D is reached, the final node embedding results are obtained and fed to *MLP* for predicting cell slew and delay. Based on the results, we calculate the loss function $\mathcal{L}_{\text{tot}}$, then leverage a gradient descent optimizer named Adam [13] to update the parameters in our work by minimizing the loss. All parameters $\boldsymbol{W}$ that need to be trained include parameters in D EdgeGAT layers $\{\boldsymbol{W}_X^d \text{ and } \boldsymbol{W}_H^d, \forall d \in \{1, ..., D\}\}$ and parameters in *MLP* $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. More importantly, all the parameters in the work can be trained end-to-end. *MLP* is implemented with 3 hidden layers.

However, the training process is time-consuming. We leverage a parallel training scheme by partitioning critical paths over multi-GPUs. Each GPU processes one graph in our parallel framework with a complete and dependent edge representation matrix and node representation matrix. Similar to the training process, we achieve parallel inference to speedup predicting process on critical paths.

Algorithm 1 Training Methodology.

Input: Edge-featured graph: $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \boldsymbol{X}, \boldsymbol{H}\}$; Node feature matrix: $\boldsymbol{X}$: $\{\boldsymbol{x}_i, \forall i \in \mathcal{V}\}$; Edge feature matrix: $\boldsymbol{H}$: $\{\boldsymbol{h}_p, \forall p \in \mathcal{E}\}$; Real PBA cell slew $S_{\text{r}}^{\text{PBA}}$ and delay $D_{\text{r}}^{\text{PBA}}$; Search depth D=100; Parameters in the LeakyReLU nonlinear function.

Output: Trainable parameters $\boldsymbol{W}$: $\{\boldsymbol{W}_X^d$ and $\boldsymbol{W}_H^d, \forall d \in \{1, ..., D\}\}$ in EdgeGAT layers; θ and ϕ in *MLP*

1: **for** $d \leftarrow 1$ to D **do**
2:   **for** $i \in \mathcal{V}$ **do**
3:     $l^d \leftarrow \log(1/d + 1)$
4:     $\boldsymbol{n}_i^d \leftarrow ((1 - l^d)\boldsymbol{I} + l^d \boldsymbol{W}_X^d)\boldsymbol{x}_i^{d-1}$,
5:     $\boldsymbol{e}_p^d \leftarrow ((1 - l^d)\boldsymbol{I} + l^d \boldsymbol{W}_H^d)\boldsymbol{h}_p^{d-1}$; ▷ Transformer
6:     Compute α_{ij}^d via Equation (2), $j \in \mathcal{N}_i$;
7:     $\boldsymbol{g}_i^d \leftarrow \sum_{j \in \mathcal{N}_i} \alpha_{ij} \boldsymbol{n}_j^d$ ▷ Node Attention Aggregator
8:     Compute β_{pq}^d via Equation (4), $q \in \mathcal{N}_p$;
9:     $\boldsymbol{z}_p^d \leftarrow \sum_{q \in \mathcal{N}_p} \beta_{pq} \boldsymbol{e}_q^d$ ▷ Edge Attention Aggregator
10:     Compute $\boldsymbol{x}_i^d$, $\boldsymbol{h}_p^d$ and $\boldsymbol{m}_i^d$ via Equation (6); ▷ Encoder
11:   **end for**
12: **end for**
13: **for** $i \in \mathcal{V}$ **do**
14:   $f_i \leftarrow \|_{d=1}^{D} \left(\boldsymbol{m}_i^d\right)$; ▷ Node embedding
15:   $S_{\text{i}}^{\text{PBA}} \leftarrow MLP(\boldsymbol{\theta} \mid \boldsymbol{F})$; ▷ Predicting cell slew
16:   $D_{\text{i}}^{\text{PBA}} \leftarrow MLP(\boldsymbol{\phi} \mid \boldsymbol{F}, S_{\text{i}}^{\text{PBA}})$; ▷ Predicting cell delay
17: **end for**
18: Compute $\mathcal{L}_{\text{tot}}$ via Equation (10);
19: Minimize $\mathcal{L}_{\text{tot}}$ via Adam [13] and update all parameters $\boldsymbol{W}$

Table 2: Benchmark statistics.

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

Table 3: Cell slew/delay prediction accuracy (R^2 score)

Benchmark	Cell Slew/Delay Prediction Accuracy (R^2 score)					
	MLP	GCNII[12]	GraphSage[7]	GAT[14]	EGNN[15]	Deep EdgeGAT
WB_DMA	0.795/0.761	0.875/0.861	0.881/0.846	0.883/0.876	0.915/0.907	**0.996/0.971**
LDPC	0.762/0.732	0.842/0.832	0.865/0.814	0.877/0.871	0.921/0.916	**0.991/0.987**
DES_PERT	0.766/0.727	0.896/0.887	0.847/0.826	0.906/0.900	0.963/0.960	**0.989/0.987**
AES-128	0.731/0.712	0.801/0.792	0.821/0.810	0.856/0.816	0.938/0.921	**0.977/0.970**
TV_CORE	0.756/0.717	0.838/0.817	0.847/0.837	0.856/0.844	0.957/0.944	**0.982/0.979**
NOVA	0.725/0.718	0.826/0.812	0.824/0.818	0.864/0.855	0.905/0.871	**0.974/0.971**
OPENGFX	0.699/0.681	0.819/0.802	0.809/0.798	0.834/0.816	0.862/0.840	**0.982/0.974**
Average	0.748/0.721	0.843/0.829	0.842/0.821	0.868/0.854	0.923/0.909	**0.984/0.977**

3.5 Path arrival Time Calculation

PBA arrival time of a critical path $AT_{\text{CP}}^{\text{PBA}}$ is estimated by cumulative addition of the predicted PBA cell delay $D_{\text{cell}}^{\text{PBA}}$ using trained model and GBA wire delay $D_{\text{wire}}^{\text{GBA}}$ from GBA results on it. $\mathcal{V}_{\text{CP}}$ and $\mathcal{E}_{\text{CP}}$ are the cell (node) and wire (edge) sets on the critical path.

$$AT_{\text{CP}}^{\text{PBA}} = \sum_{\text{i} \in \mathcal{V}_{\text{CP}}} D_{\text{i}}^{\text{PBA}} + \sum_{\text{p} \in \mathcal{E}_{\text{CP}}} D_{\text{p}}^{\text{GBA}}. \tag{11}$$

According to our experimental results, the path delay pessimism in GBA is reduced obviously without significant runtime overhead.

4 Experimental Results

We implement our models using PyTorch and develop deep EdgeGAT based on the proposed graph learning framework. 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. PBA and GBA are performed using PrimeTime [1] on a 72-core 2.6GHz Linux machine with 1024 GB memory. We use R^2 score [10] to evaluate the relative cell slew/delay and path arrival time accuracy on the testing benchmarks. The maximum absolute error of path arrival time for different designs is reported.

In this work, we train and evaluate deep EdgeGAT model based predictors using open-source designs. Specifically, we generated PBA and GBA timing reports for 18 open-source circuits with TSMC28nm technology using STA tool PrimeTime [1] while the timing constraint is set to 0.5ns (The value of clock period). The critical paths (CPs) in open-source circuits are split into training and testing cases as shown in Table 2.

Table 4: Path arrival time prediction accuracy based on GBA timing results, including R^2 score / MAE (ps), and runtime (s) comparison. "MAE" represents maximum absolute error and D represents number of EdgeGAT layers used.

Benchmark	Path Delay Prediction Accuracy: R^2 score / MAE(ps)						Runtime(s)				
	Commercial STA Tool [1]		Prior Work	Ours			PBA		Ours		Comparison
	PBA	GBA	CART [6]	D=25	D=50	D=100	Full	GBA	Predictor	Total	Speedup
WB_DMA	1.000/0.00	0.549/64.91	0.732/21.34	0.881/10.74	0.928/3.23	**0.998/0.89**	276.7	12.1	**1.197**	13.297	**20.81×**
PCI_BRIDGE	1.000/0.00	0.471/89.23	0.694/41.01	0.896/14.65	0.901/9.51	**0.993/1.46**	365.9	15.3	**0.798**	16.098	**22.73×**
DES_PERT	1.000/0.00	0.452/50.84	0.702/37.86	0.891/25.17	0.931/10.92	**0.997/1.02**	386.3	16.4	**1.614**	18.014	**21.44×**
AES-256	1.000/0.00	0.393/130.92	0.511/80.75	0.702/22.94	0.822/9.37	**0.977/3.94**	593.7	31.2	**2.731**	33.931	**17.50×**
TV_CORE	1.000/0.00	0.424/91.27	0.651/57.93	0.825/29.36	0.897/19.34	**0.984/6.81**	614.6	22.1	**2.410**	24.51	**25.08×**
NOVA	1.000/0.00	0.419/88.64	0.673/36.59	0.839/23.83	0.904/14.37	**0.983/4.11**	1133.8	30.5	**4.276**	34.776	**32.60×**
OPENGFX	1.000/0.00	0.378/267.91	0.571/147.03	0.793/53.74	0.851/27.89	**0.987/5.84**	1185.4	36.3	**4.432**	40.732	**29.10×**
Average	1.000/0.00	0.441/111.96	0.647/60.36	0.832/25.78	0.891/13.52	**0.988/3.44**	642.3	23.4	**2.494**	25.894	**24.80×**

4.1 Cell Slew and Delay Prediction Accuracy

We compare the cell slew and delay prediction performance based on deep EdgeGAT with the state-of-the-art classical graph learning methods, including *MLP* without GNN, GCNII [12], GrapgSage [7], GAT [14] and EGNN [15]. Note that we overcome the over-smoothing issue in these models like our work. Open-source cases in Table 2 are used for training and testing. All graph learning models are used to perform node embedding with search depth D=100, and *MLP* modules are used to predict PBA cell slew/delay based on different node embedding results.

Table 3 shows all the cell slew/delay prediction accuracy (R^2 score) results of different methods. Compared with using the graph learning method, the accuracy of *MLP* predictor is extremely limited. For the rest work, the performance of the proposed method is significantly better than other graph learning methods. The average R^2 scores of deep EdgeGAT which represent cell slew and delay prediction accuracy reach 0.984 and 0.977, which outperforms GCNII by 0.142/0.147, GraphSage by 0.141/0.156, and GAT by 0.116/0.123. Since they consider only node features and path structure information, these methods suffer bottlenecks for post-layout designs with limited learning ability. Compared with EGNN which also considers the edge features, our work achieves average gains of 0.062/0.069 on the seven industrial designs.

The model training progress is timing-consuming, with about 12 hours on a single GPU. However, the parallel training method on multiple GPUs achieves 6× speedup on our servers.

4.2 Path arrival Time Calculation Performance

Table 4 shows the path arrival time prediction accuracy through our workflow and conventional machine learning method proposed in [6]. Note that the baseline results of path arrival time in PBA mode are from the golden timing analysis tool, PrimeTime [1]. In Table 4, there is an obvious difference between PBA and GBA timing results. For GBA mode, the average R^2 score is 0.441 and the maximum absolute error reaches 111.96ps, which is extremely pessimistic. According to the R^2 scores, the accuracy of our work reaches 0.832, 0.891, and 0.988 on average when D=25,50 and 100. And the average maximum absolute error of our results is just 3.44ps.

More importantly, the predictor runtime costs 2.494s on average when D=100 for different designs scaling from 40k to 210k cells, as shown in Table 4. Combined with GBA timing analysis, the average runtime of our workflow to get accurate PBA timing results costs 25.894s, which achieves 24.80× speedup compared with PrimeTime.

5 Conclusion

In this work, we are the first to apply graph learning techniques to predict PBA timing results based on GBA results, addressing an important accuracy-runtime tradeoff in STA. Our work is composed of node embedding based on proposed EdgeGAT layers, predicting cell slew/delay synergistically based on node embedding results and calculating path arrival time through cumulative addition of predicted PBA cell delay values on timing path. Compared with conventional graph learning method, deep EdgeGAT can learn global path information. Experimental results on open-source designs with different scales demonstrate that our predictor is both accurate and fast. The inductive model can be shared across different designs without loss of accuracy even if they are unseen.

Acknowledgment

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

References

[1] Synopsys, "Primetime user guide," http://www.synopsys.com/Tools/Implementation/SignOff/Documents/primetime_ds.pdf, 2015.
[2] Cadence, "Tempus user guide." https://www.cadence.com/content/cadencewww/global/enUS/home/tools/digital-design-and-signoff/silicon-signoff/tempustiming-signoff-solution.html, 2015.
[3] Z. Zhang, Z. Guo, Y. Lin, R. Wang, and R. Huang, "Eventtimer: fast and accurate event-based dynamic timing analysis," in *Proc. DATE*, 2022, pp. 945–950.
[4] R. Molina, "EDA vendors should improve the runtime performance of path-based timing analysis," *Electronic Design*, vol. 20136, 2013.
[5] A. B. Kahng, "Machine learning applications in physical design: Recent results and directions," in *Proc. ISPD*, 2018, pp. 68–73.
[6] A. B. Kahng, U. Mallappa, and L. Saul, "Using machine learning to predict path-based slack from graph-based timing analysis," in *Proc. ICCD*, 2018, pp. 603–612.
[7] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Proc. NIPS*, vol. 30, 2017.
[8] 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, pp. 1–6.
[9] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin, "A timing engine inspired graph neural network model for pre-routing slack prediction," in *Proc. DAC*, 2022, pp. 1207–1212.
[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*, 2021.
[11] S. Sun, Y. Jiang, F. Yang, B. Yu, and X. Zeng, "Efficient hotspot detection via graph neural network," in *Proc. DATE*, 2022, p. 1233–1238.
[12] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proc. ICML*, 2020, pp. 1725–1735.
[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.
[15] L. Gong and Q. Cheng, "Exploiting edge features for graph neural networks," in *Proc. CVPR*, 2019, pp. 9211–9219.