INTEGRATION, the VLSI journal 99 (2024) 102262

Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

Pre-route timing prediction and optimization with graph neural network models

Kyungjoon Chang, Taewhan Kim *

Department of Electrical and Computer Engineering, Seoul National University, Republic of Korea

ARTICLE INFO

Keywords:
Physical design
Timing prediction
Machine-learning
Placement
Interconnects

ABSTRACT

In recent years, the application of deep learning (DL) models has sparked considerable interest in timing prediction within the place-and-route (P&R) flow of IC chip design. Specifically, at the pre-route stage, an accurate prediction of post-route timing is challenging due to the lack of sufficient physical information. However, achieving precise timing prediction significantly accelerates the design closure process, saving considerable time and effort. In this work, we propose pre-route timing prediction and optimization framework with graph neural network (GNN) models combined with convolution neural network (CNN). Our framework is divided into two main stages, each of which is further subdivided into smaller steps. Precisely, our GNN-driven arc delay/slew prediction model is divided into two levels: in level-1, it predicts net resistance (net R) and net capacitance (net C) using GNN while the arc length is predicted using CNN. These predictions are hierarchically passed on to level-2 where delay/slew is estimated with our GNN based prediction model. The timing optimization model utilizes the precise delay/slew predictions obtained from the GNN-driven prediction model to accurately set the path margin during the timing optimization stage. This approach effectively reduces unnecessary turn-around iterations in the commercial EDA tools. Experimental results show that by using our proposed framework in P&R, we are able to improve the pre-route prediction accuracy by 42%/36% on average on arc delay/slew, and improve timing metrics in terms of WNS, TNS, and the number of timing violation paths by 77%, 77%, and 64%, which are an increase of 32%/35% on arc delay/slew and 30%, 20% and 31% on timing optimization compared with the existing DL prediction model.

1. Introduction

Timing closure is an essential task in integrated circuit (IC) design aimed at fulfilling the timing requirements of the design. However, this task demands significant time and effort, and may lead to additional costs and overhead. Achieving timing closure involves an intricate optimization process, requiting to address various factors. To reduce the overhead on timing closure, the pre-route stage on the IC design flow involves operations such as gate sizing, buffer insertion, and gate displacement. These optimizations not only improve timing closure but also enhance design convergence. One of the main challenges in these design optimizations at the pre-route stage is the lack of physical data related to timing, thereby relying on estimation on the signal delays and transition times for the unrouted nets. Incorrect pre-route timing prediction hinders the optimization process, necessitating long design iterations to converge the design implementation. Fig. 1 illustrates this situation in which the flow proceeds based on the predicted values after placement. However, if the prediction is not so accurate, it leads to an inability to achieve timing closure, necessitating a return to the previous stage to reinitiate the flow.

Moreover, as the process technology node shrinks 7 nm and below [1], the portion of interconnect delay is becoming large compared to previous nodes, its importance is being increasingly recognized. Therefore, it is crucial not only to predict cell delays but also to accurately predict net delays. If a substantial portion of net delays is inaccurately predicted, it could lead to erroneous estimation of critical paths and ultimately results in an improper timing optimization, which causes an necessarily longer design iteration.

Fig. 2 illustrates the correlation between the values of timing parameters (pin-to-pin delay and net resistance (R)) before and after routing. The result shows a maximum difference of 372ps in arc delay and 8 kΩ in net resistance, indicating a significant variance between the predicted values before routing and the actual measurements after routing. This implies that an accurate pre-route prediction is very important to make a fast convergence on timing closure.

In recent years, several deep-learning (DL) models have been developed and proposed to enhance post-route design results at the pre-route stage. Prediction targets other than timing include IR drop (e.g., [2]),

* Corresponding author.
E-mail addresses: kyungjoon0@snucad.snu.ac.kr (K. Chang), tkim@snucad.snu.ac.kr (T. Kim).

https://doi.org/10.1016/j.vlsi.2024.102262
Received 16 October 2023; Received in revised form 5 May 2024; Accepted 13 August 2024
Available online 19 August 2024

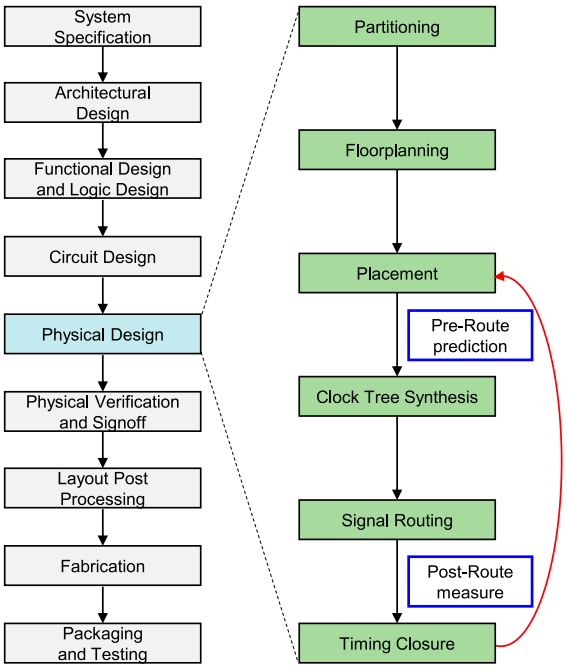

Fig. 1. The physical design flow and the specific processes.

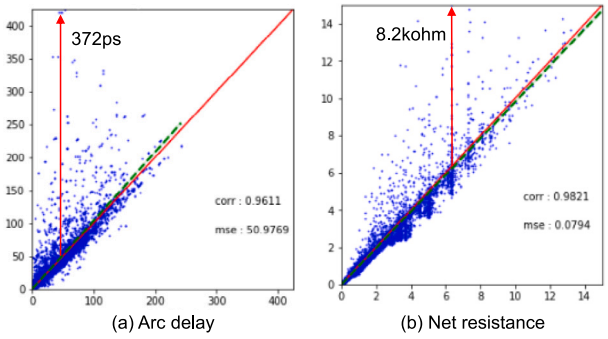

Fig. 2. Correlation between the estimated values at the placement and the actual values related to the arc delay in (a) and net resistance in (b) of the implementation of benchmark `ldpc` using Nangate 15 nm process design kit (PDK).

design rule check (DRC) hotspots (e.g., [3,4]) and routability (e.g., [5]). In addition, some works (e.g., [2,3]) enhance their design results by further optimizing with prediction results. Several DL based models have been proposed to enhance the accuracy of timing prediction. The works in [6–8], predicted the sign-off timing by analyzing the post-route circuit. These models employed relevant data extracted from P&R tools [6], sign-off timing tools [7], and signal integrity-induced delay increments [8] to predict the sign-off wire delay and slew. These studies served as pioneering efforts in using DL to predict timing, but they focused on post-route timing prediction rather than pre-route timing prediction. For this reason, research on predicting timing at the pre-route stage is relatively scarce. Recently, there has been a study that predicts post-route timing at the pre-route stage and incorporates it into commercial tools. The work in [9] partitioned nets into multiple zones based on their length and fanout. Then, they utilized linear regression to derive an R/C scaling factor to establish a correlation between pre-route and post-route resistances and capacitances for each respective zone. Then, the R/C values used in a commercial EDA tool are replaced with the scaled values in a lookup table format to improve the accuracy of timing prediction. However, this study is limited to predicting R and C associated with timing, and it raises concerns about its generality since it focuses on predicting the same design after being trained on it once. On the other hand, the work in [10] predicted arc delay and arc output slew from net features extracted at the pre-route stage, which is then expanded to crosstalk prediction [11]. However, this study is primarily focused on crosstalk-related research and lacks sufficient features related to the context with the surrounding components. Moreover, it lacks essential information on R and C, which plays a crucial role in timing. Furthermore, in addition to its imprecise method for path delay calculation based on PERT traversal [12], it solely compares the predicted timing results without providing any optimization approaches. On the contrary, the work in [13] extracted multiple features for arcs and utilized then on multi layer perceptron(MLP) and CNN models considering routing congestion to predict delay and slew. This approach led to significant advancements and notable improvements compared to previous research. However, the contribution of the CNN model to the results was found to be minimal, and the MLP model lacked sufficient capability to represent the relationships with the surroundings. In particular, the prediction of the outliers which are detour nets is not so accurate.

Recently, some other networks such as graph neural network (GNN) and recurrent neural network (RNN) have been introduced to improve prediction accuracy of timing and classifying critical path. The work in [14], proposed a method for pre-route arrival time and slack prediction which used GNN at timing endpoints instead of static timing analysis (STA) tools. The work in [15] proposed GNN based static timing analysis to predict cell and slew delay as well as path delay prediction. Furthermore, the work in [16] proposed a fast and accurate wire timing estimation based on customized GNN by considering neighboring gates. However, most of the GNN based timing-related tasks focused on predicting timing metrics. There has been a notable absence of study investigating the impact of incorporating the predicted timing into commercial tools on timing closure. The study presented in [17] employed a long-short term memory (LSTM) network which is a type of RNN based on graph learning to predict post-routing total negative slack (TNS) during the early placement and clock tree synthesis (CTS) stages of the sequential flow. Critical path prediction method has been proposed with BiLSTM in [18] to alleviate the overhead of long design iteration. It achieved an accurate classification of critical paths compared to the previous studies by considering both fan-in and fan-out gates, not just fan-in alone. Indeed, accurately distinguishing the critical paths aids in reducing the long design iteration. However, since it cannot precisely predict the path delays, it could not significantly contribute to reducing long design iteration. Despite the existence of numerous machine learning models for path delay prediction, there is currently no research that employs GNN models for both prediction and integrating them into the P&R process for implementation.

To overcome the limitations of prior works, we introduce a new approach to predicting path delay. It utilizes a sophisticated combined model of GNN and CNN to incorporate the correlation with neighboring arc delays and meticulous consideration of routing congestion. Furthermore, our proposed framework involves the utilization of predicted delays within a commercial tool for implementation, thereby facilitating the rapid achievement of timing closure requirements. The main contributions of our work can be summarized as follows:

- We proposed a net and arc GNN model, which allows the reflection of relationships with the surrounding elements unlike previous research that solely compared values of multiple features and routing congestion. In addition, by reflecting features of routing resources we achieved a very accurate prediction of net R and C as well as arc delay and slew.
- We propose a novel deep-learning model which hierarchically connects two levels of prediction for higher delay prediction accuracy. In the initial stage, which is a combined model of GNN and CNN, predictions are made for net R, net C, and arc length, which form the foundation for delay and slew estimation. Subsequently,

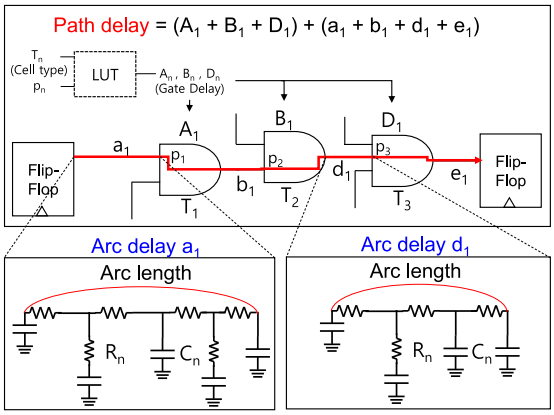

Fig. 3. Relation among path delay, arc delay, and arc slew.

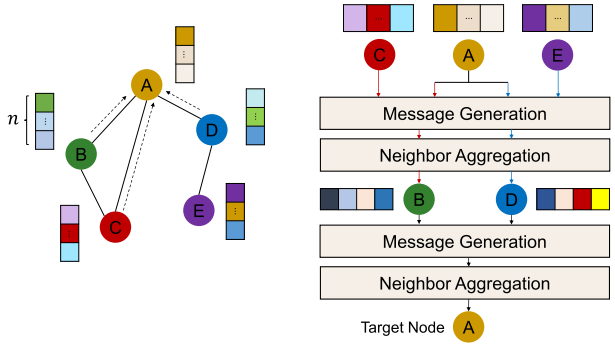

Fig. 4. Illustration of graph neural network.

the more refined predictions from the previous step are passed on to the next stage, enabling the use of more sophisticated features for delay prediction through another GNN.

- We propose improved annotation method considering design characteristics which fully implement accurate prediction values into EDA tools. Consequently, this reduces unnecessary additional design iterations in design optimization and facilitates faster convergence in timing closure.

The rest of the paper is organized as follows. Section 2 provides related preliminaries and Section 3 describes our proposed prediction framework, followed by the implementation of our prediction delay into commercial optimization tools in Section 4. Then, experimental results and discussion are given in Section 5. Finally, Section 6 concludes this work.

2. Preliminaries

2.1. Inter-relation among timing parameters

Fig. 3 shows the example of inter-dependency among the timing parameters commonly used in commercial EDA tools to calculate path delay. At the pre-route stage, the primary parameter we aim to estimate is the path delay, which is determined by the summation of gate delays and arc delays along the path. As depicted at the top in Fig. 3, the path delay is composed of the summation of gate delays (A_1, B_1, D_1) and arc delays (a_1, b_1, d_1, e_1). Since, practically, the gate delays are derived from a form of look-up table that uses combinations of gate input slew and output load capacitance to predict a non-linear gate delay, we aim to accurately predict the gate input slew or arc output slew. Each arc delay can be expressed as the product and sum of the R and C values for each segment by following the Elmore delay [19] formulation. Additionally, it is natural for arc delay to increase proportionally with length. Hence, the arc length also influences the calculation of arc delay. Therefore, by prioritizing the accurate prediction of R, C, and arc length, and utilizing these along with additional features to predict arc delays and arc output slews precisely, one is able to ultimately achieve an accurate estimation on the path delay.

2.2. Graph attention network

Graph Neural Network (GNN) [20] is a machine learning technique designed to process graph-structured data. Graphs in GNN consist of nodes and edges, representing complex data structures where nodes correspond to entities and edges indicate relationships or interactions between these entities. GNN takes graphs as input and models the interactions between nodes to learn and infer useful information from the graph. Fig. 4 illustrates an example of the operation of a traditional GNN consists of two layers. Considering the target node A, its adjacent nodes are B and D while each of B and D has adjacent nodes (A, C) and (A, E), respectively. Messages are received and aggregated from each adjacent node to create a vector representation for the target node A. Graph Convolutional Network (GCN) [21] is one of the most fundamental models representing the GNN, as described above. Graph Attention Network (GAT) [22] is another type of GNN that aims to enhance the representation learning process in graph-structured data. In contrast to GCN in which the importance of neighboring nodes was treated equally, GAT introduces attention mechanisms to effectively weigh the importance of neighboring nodes during message passing. In GAT, each node in the graph maintains a feature vector, and attention coefficients are learned for each edge connecting the node to its neighbors. These attention coefficients determine the importance or relevance of each neighbor's information in relation to the central node. By assigning higher attention weights to more relevant neighbors and lower weights to less relevant ones, GAT can selectively emphasize significant nodes during the aggregation of information. In addition, the multi-head attention mechanism is an extension of the standard attention mechanism and allows the model to attend to different parts of the input simultaneously. This approach enhances the model's ability to capture diverse patterns and relationships in the graph data.

Fig. 5 illustrates the multi-head attention where the target node A receives three attention messages from its adjacent neighboring nodes and three self-attention messages. By computing and aggregating information from various sources, the model leverages complementary insights to enhance its generality. This approach enables the model to effectively combine diverse information, contributing to improved performance and adaptability in different contexts. Therefore, the attention mechanism in GAT enables the network to adaptively focus on different parts of the graph, thus allowing the model to capture complex relationships and dependencies effectively.

2.3. Convolution neural network

A Convolutional Neural Network (CNN) [23] is a deep learning model primarily designed for processing grid-like data such as images and time series. It has emerged as a powerful technique for various computer vision tasks and has shown exceptional performance in a wide range of applications. The fundamental building blocks of CNN are convolutional layers which perform a process called convolution. Convolution involves applying a set of learnable filters, also known as kernels, to the input data. These filters detect different features, patterns, or edges in the data and generate feature maps as output. Through these convolution blocks, CNN is able to effectively identify significant patterns and features in the input, resulting in enhanced performance over the conventional method of multi layer perceptron (MLP). Fig. 6 illustrates the fundamental structure of CNN where

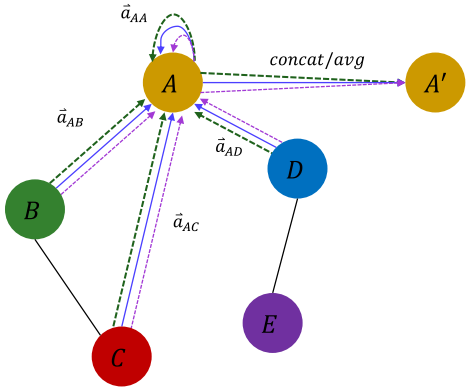

Fig. 5. Multi-head attention of graph attention network [22].

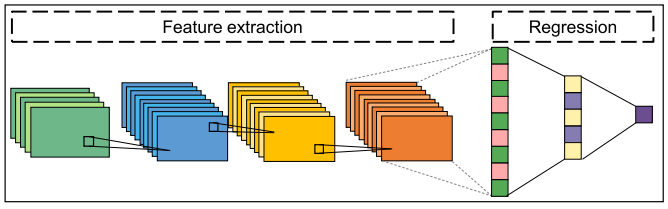

Fig. 6. Construction of convolutional neural network.

multiple convolutional layers are used to extract features from the input data. These extracted features are then utilized in a regression process to obtain numerical values or vector.

2.4. XGBoost

XGBoost [24], which stands for eXtreme Gradient Boosting, is a highly efficient and scalable implementation of gradient boosting, a machine learning technique. It has gained significant attention because of its performance in various predictive modeling competitions and real-world applications. Gradient Boosting Machine (GBM) is a machine learning technique for regression and classification problems. It builds an ensemble model by sequentially adding predictors (typically decision trees) where each successive predictor corrects its predecessor. XGBoost has been successfully applied in various application domains, including in predicting performance in circuit design. Its ability to handle a variety of data types and distributions makes it versatile for both regression and classification tasks.

3. Proposed timing prediction methodology

3.1. Overall flow

Our proposed pre-route timing prediction and optimization framework is depicted in Fig. 7. The framework has two main flows. One flow, indicated by the blue arrows, uses our proposed ML based model for timing training and prediction. The other flow, indicated by red arrow, carries out timing optimization by utilizing a conventional EDA flow through annotating internally the predicted timing values obtained in the timing prediction flow. In the prediction step, the timing prediction is accomplished by using a combination of GNN and CNN models. This flow involves predicting various timing parameters such as net R, net C, arc length, arc delay, and arc slew to achieve precise timing estimation. In order to train the ML models, we extract features and labels from the prepared circuits. In the optimization step, the timing prediction values obtained from the ML models are incorporated into a commercial EDA tool. This integration enables the

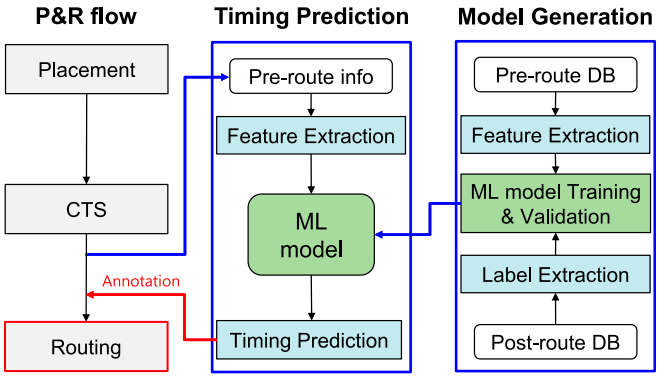

Fig. 7. The overall flow of our proposed timing prediction(blue line) and optimization(red line) framework.

EDA tool to leverage the accurate timing information during the design optimization process. With a more accurate predicted timing, the EDA tool is able to perform design optimization more effectively and refine the implementation to meet the timing requirements.

In the following subsections, we describe our proposed ML based timing prediction, followed by describing our timing optimization.

3.2. Structure of prediction model

Fig. 8 and Table 1 illustrates the comprehensive architecture of the machine learning-based pre-route timing prediction model and its features. This structure is divided into two main stages and comprises a total of four models. In the initial stage, referred to as level-1, three models are employed. These models respectively predict net R and net C using GNN, and predict arc length increments using CNN. The predicted net R, net C which are net information and delta arc length which represent congestion information are further incorporated as input features for level-2, which focuses on predicting arc delays and arc output slews. Particularly at level-2, the arc output slew predicted at level-2 replaces the arc output slew of the input features for arc delay. The architecture encompasses a hierarchical approach where level-1 predictions inform and enhance the input features for level-2 predictions. This structure enables the model to effectively capture the intricate timing dependencies within the circuit design.

3.3. GNN prediction models for net R and C

Our net R and net C prediction models are structured using GNN as depicted in the upper portion in level-1 in Fig. 8. Both the net R and net C models have their independent features, while sharing common features. Those features are described below:

- **Net R, Net C:** These features extracted from pre-route parasitic report files which are the predictions of EDA tool.
- **Net length:** It corresponds to the length of the net, which is taken from the parasitic report files of EDA tool.
- **Fanout:** It indicates the number of net sinks.
- **Net RUDY:** Estimated congestion values of the window.
- **Max/Min Via layer:** It is taken from pre-route parasitic report, which is predictive information regarding which layer to use.
- **Net # vias:** It corresponds to the predictive number of vias which net will use from parasitic report.

Net RUDY [12] quantifies congestion in a net, its value is calculated using the following equation.

$$net\_RUDY(i) = \frac{Vol(i) + \sum_{j \in N(i)} [Vol(j) \times OL(i,j)/Area(j)]}{Area(i)} \quad (1)$$

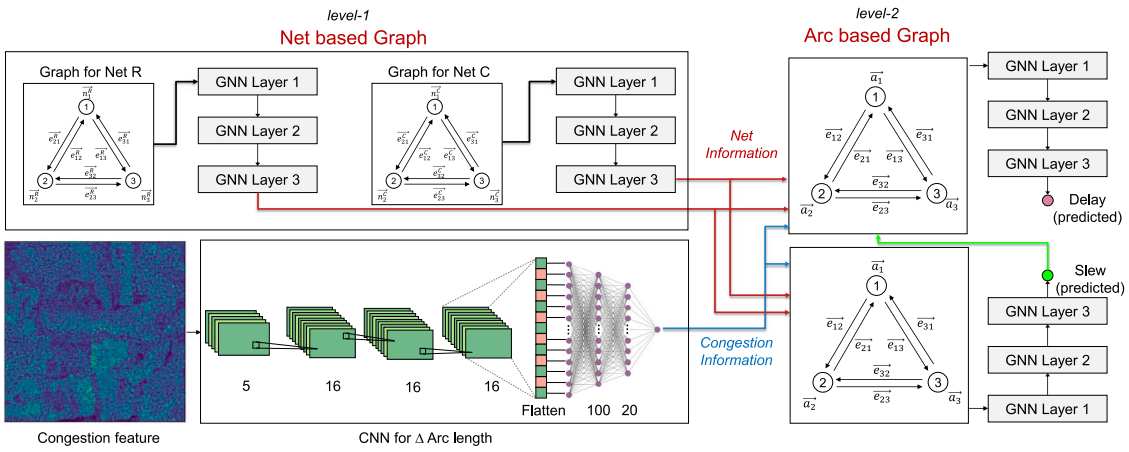

Fig. 8. Architecture of our pre-route timing prediction framework, which predicts net R, net C, and arc length with three sub-models in *level-1* (left part), and predicts arc delay, arc slew in *level-2* (right part).

Table 1
Summary of input features for each of our sub-models.

Feature	Description	Data usage
Arc delay	Source to sink propagation delay	Level-2 delay/slew
Arc input slew	Slew value at arc input	
Arc output slew	Slew value at sink pin	
Pin-to-pin distance	Source-to-sink Manhattan distance	
Arc R	Cumulative resistance from source to sink.	
τ(Arc RC)	Elmore RC delay from source to sink.	
Arc max via	Highest via layer from source to sink.	
Arc min via	Lowest via layer from source to sink.	
Net R	Total resistance of Net	Level-1 R
Net C	Total capacitance of Net	Level-1 C
Net length	Total wire length of Net	Level-1 R, C
Fanout	Number of net sinks	
Net Rudy	Estimated congestion values in a net bounding box	
Net max via	Highest via layer of Net	
Net min via	Lowest via layer of Net	
Net # of via	Number of via of Net	
G-cell RUDY map	Estimated congestion values on G-cell	Level-1 congestion
Horizontal net density map	Horizontal track usage value on G-cell	
Vertical net density map	Vertical track usage value on G-cell	
Source & sink location map	Location of arc source and sink on G-cell	
Pincap location map	Cumulative pin capacitance on G-cell	

where $Area(i)$, $Vol(i)$, $N(i)$, and $OL(i, j)$ indicate the bounding box area of net i, wire volume of net i, the overlapped net of net i, and the overlapped area of nets i and j, respectively. Net RUDY is a metric used to quantify congestion within a net. It assesses the level of uniformity in wire density across the routing resources that a net traverses. In other words, it measures how evenly the wires are distributed within the area occupied by the net. A higher net RUDY value indicates a heavier congestion, implying that the net's wires are densely packed, possibly causing slower signal propagation and potential timing issue. However, representing the congestion of the surrounding area with a single value indices a limitation. Note that a set of novel features are introduced in our study, particularly those related to vias, which are the via count and the highest/lowest layer of vias. Those features provide information about the lowest and highest layers of vias that a net uses or will use. This information is intended to supply routing information before routing. These supplementary features along with RUDY [12] which quantifies congestion provides a more comprehensive representation. This combination of features not only refines the representation of congestion but also offers anticipatory information about the specific areas where routing resources will be deployed.

Training our net R and net C prediction sub-models is conducted separately. In this process, the seven input features traverse three GNN layers, eventually converging into a 1-bit output. This output corresponds to either the post-route net R or net C values. These trained sub-models will subsequently be utilized in the training phase of the arc delay/slew prediction at the *level-2*. The rationale behind utilizing three GNN layers can be elucidated through an example. Fig. 9 provides an illustration of the GNN layers and the receptive field around a target node. In this scenario, the red node is the target node and the general graph structure is depicted in Fig. 9(a). When using a total of three GNN layers, the scope of the first layer encompasses the blue nodes, the second layer extends to the green nodes, and finally the third layer covers the yellow nodes, effectively defining the receptive field. However, if the number of GNN layers becomes excessively large, local information might be lost, and the entire graph would function as a receptive field. Conversely, if the number of GNN layers is too small, it may not effectively capture the information from the neighboring nodes. Consequently, through the experiments with net R and net C prediction sub-models, it has been determined that employing three GNN layers is the most effective.

One model component of our proposed framework, net based graph, represents the interconnections between different nets in the circuit. Each node in this graph corresponds to a net and the edges indicate the connections between nets. As seen in Fig. 8, the Net-based graph is divided into two types, which are as follows:

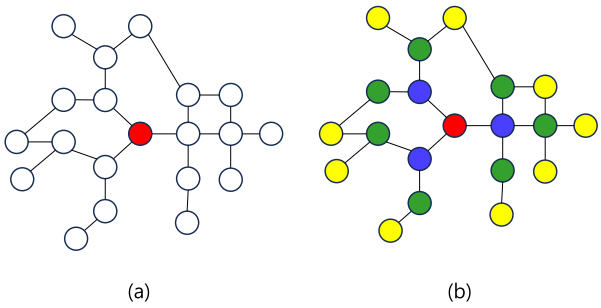

Fig. 9. Example of inter-relation between GNN layer and receptive field of the target node (red).

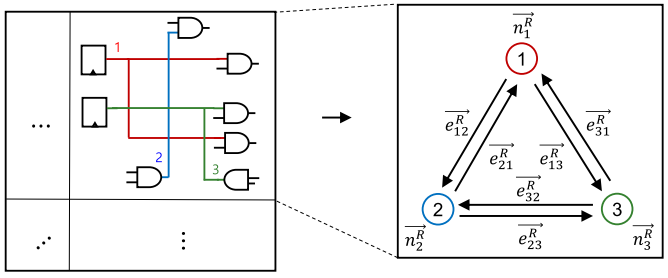

Fig. 10. Generation of net based graph from circuit.

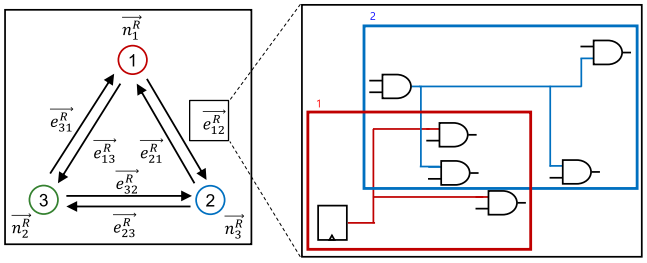

Fig. 11. Generation of edge feature from graph.

3.3.1. Net based graph for R

We partitioned the entire design layout into n by n grid. For each grid, in the net-based graph $G_R(V_R, E_R)$, each node $n_i^R \in V_R$ corresponds to a net $n_i^R \in N$, and the node feature of $\overrightarrow{n_i^R}$ is concatenated vectors of level-1 R features. For example, Net R graph as depicted in Fig. 10, the node $\overrightarrow{n_i^R}$ for Net R can be represented as [R, NL, F, NR, NMaxV, NMinV, #V], where each element represents the resistance of the net, net length, fanout of the net, net Rudy, highest via layer of net, lowest via layer of net, and the number of vias, respectively. For each net, we draw the bounding box according to the coordinates of its source and sinks. Subsequently, for nets whose bounding boxes overlap, we connect the corresponding nodes with edges. Each edge $e_{ij}^R \in E$ is a directed edge from $n_j^R \in V_R$ to $n_i^R \in V_R$, representing whether there is an overlap between n_i^R and n_j^R. The edge feature $\overrightarrow{e_{ij}^R}$ of e_{ij}^R can be expressed as $\overrightarrow{e_{ij}^R} = (avgx_i, avgy_i, avgx_j, avgy_j, RP_{n_i^R,n_j^R})$, where each component represents the average x and y coordinates of the source and sink of n_i^R and n_j^R, and $RP_{n_i^R,n_j^R}$ denotes the relative position of n_j with respect to n_i^R. Specifically, $RP_{n_i^R,n_j^R}$ is encoded as follows: if n_j^R is positioned to the right-upper corner with respect to n_i, it is denoted as 1; if n_j^R is positioned to the left-upper corner, it is denoted as 2; if n_j^R is positioned to the left-lower corner, it is denoted as 3; and if n_j^R is positioned to the right-lower corner, it is denoted as 4. For instance, in the case of n_1 from Fig. 11, if the coordinates of its source and two sinks are (0, 0), (3, 1), and (2, 3), respectively, then the average x and y coordinates of n_1^R are 1.67 and 1.34. If we consider n_2^R with its source and three sinks having coordinates (1, 4), (2, 2), (4, 2), and (5, 5), respectively, then the average x and y coordinates of n_2^R are 3 and 3.25. Looking at the average x and y coordinates of each, we can determine that n_2^R is in the first quadrant with respect to n_1^R, so $RP_{n_1^R,n_2^R}$ is 1. Conversely, n_1^R is in the third quadrant with respect to n_2^R, so $RP_{n_1^R,n_2^R}$ is 3. Thus, we can represent $\overrightarrow{e_{12}^R} = [1.67, 1.34, 3, 3.25, 1]$, and $\overrightarrow{e_{21}^R} = [3, 3.25, 1.67, 1.34, 3]$.

3.3.2. Net based graph for C

We partitioned the entire design layout into n by n grid. For each grid in the net-based graph $G_C(V_C, E_C)$, every node $n_i^C \in V_C$ corresponds to a net $n_i^C \in N$, and the node feature of $\overrightarrow{n_i^C}$ consists of concatenated vectors of level-1 R features. For instance, the node $\overrightarrow{n_i^C}$ representing Net C can be characterized as [C, NL, F, NR, NMaxV, NMinV, #V], which is identical to the features of Net R except for the 'C' component. Similar to the edges in Net R, each edge $e_{ij}^C \in E$ is a directed edge from $n_j^C \in V_C$ to $n_i^C \in V_C$ signifying whether there is an overlap between n_i^C and n_j^C. The edge feature $\overrightarrow{e_{ij}^C}$ of e_{ij}^C can be defined as $\overrightarrow{e_{ij}^C} = (avgx_i, avgy_i, avgx_j, avgy_j, RP_{n_i^C,n_j^C})$

Creating a graph through the above method and updating node features by passing it through a single GAT layer considering the information of neighboring nodes. The graph attention layer is a layer that generates node embeddings based on self- attention. The output $\mathbf{h}_i^{(k)}$ of the kth graph attention layer in GAT is defined as follows:

$$s_i^{(k)} = \alpha_{i,i}^{(k)} \mathbf{W}^{(k)} \mathbf{h}_i^{(k-1)} + \sum_{j \in N(i)} \alpha_{i,j}^{(k)} \mathbf{W}^{(k)} \mathbf{h}_j^{(k-1)} \tag{2}$$

$$\mathbf{h}_i^{(k)} = f_k(s_i^{(k)}) \tag{3}$$

where $N(i)$, $\mathbf{W}^{(k)}$, f_k and $\mathbf{h}_i^{(k-1)}$ represent the set of neighbor indices for node i, the weight matrix for the kth graph attention layer, activation function and is the output of the previous graph attention layer, with the initial input node feature represented as $\mathbf{h}_i^{(0)} = x_i$ respectively. Furthermore, the attention score $\alpha_{i,j}^{(k)}$ in Eq. (2) is defined as Eq. (4):

$$\alpha_{i,j}^{(k)} = \frac{exp(\phi^{(k)}(\mathbf{W}^{(k)}\mathbf{h}_i^{(k)}, \mathbf{W}^{(k)}\mathbf{h}_j^{(k)}))}{\sum j \in N(i) exp(\phi^{(k)}(\mathbf{W}^{(k)}\mathbf{h}_i^{(k)}, \mathbf{W}^{(k)}\mathbf{h}_r^{(k)}))} \tag{4}$$

In Eq. (4), $\phi^{(k)} : Z \times Z \rightarrow R$ denotes the attention function of the kth graph attention layer, which is based on a Leaky ReLU function and defined according to Eq. (5):

$$\phi^{(k)}(\mathbf{W}^{(k)}\mathbf{h}_i^{(k)}, \mathbf{W}^{(k)}\mathbf{h}_j^{(k)}) = LeakyReLU(a_k^T[\mathbf{W}^{(k)}\mathbf{h}_i^{(k)} \oplus \mathbf{W}^{(k)}\mathbf{h}_j^{(k)}]) \tag{5}$$

where a_k^T is a trainable weight vector and $\oplus$ symbolizes vector concatenation. In summary, the graph attention layer creates new node embeddings $\mathbf{h}_i^{(k)}$ through a process delineated by Eqs. (2) to (5), leveraging a trainable weight matrix $\mathbf{W}^{(k)}$ and the weight vector a_k.

3.4. CNN model for congestion prediction

Our congestion prediction model which follows the CNN model used in [13], bottom located in *level-1* in Fig. 8, employs CNN to predict the variations in arc length that occurs due to detours in congested areas. While theoretically arc length should only increase if there is a significant cell movement, our investigation found some instances where the positioning of routed pins could lead to a slight decrease in arc length. For this reason, our research emphasizes the prediction in arc length magnitude within the congestion model, given that the congestion in a specific area correlates closely with the conditions in adjacent regions.

The rationale behind utilizing CNN for the congestion model stems from its adeptness at capturing spatial aspects. To construct CNN

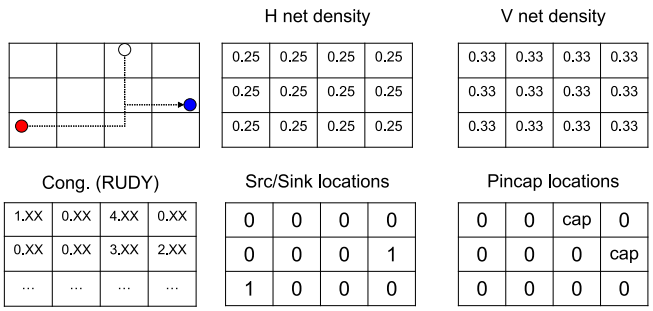

Fig. 12. Example of input feature extraction for a target arc (red→blue).

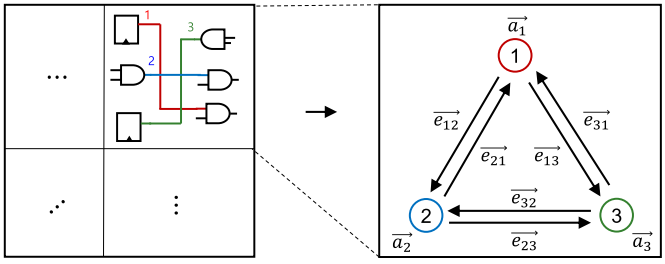

Fig. 13. Generation of arc based graph from circuit.

features, we partitioned the entire design layout into G-cells. For each G-cell, a 3D array with dimensions of $5\times W\times H$ was established, serving as input channels for each net. Here, W and H denote the width and height of the net bounding box, respectively, measured in terms of G-cell count. The five features entering as input channels are the G-cell RUDY map, horizontal and vertical net density map [3], source/sink location, and pin capacitance location. These features are described below:

- **G-cell RUDY map:** It indicates an estimated congestion value with RUDY on a G-cell.
- **Net density map:** It indicates an estimated horizontal and vertical track usage value on a G-cell.
- **Source/sink location:** It indicates the locations of the arc source and sink on a G-Cell.
- **Pin capacitance location map:** It indicates an accumulated pin capacitance value and its location on a G-cell.

Fig. 12 shows an example depicting the schematic of a net concerning the target net and its corresponding input features. The G-cell RUDY holds RUDY values for each of the $W\times H$ G-cells, while the horizontal/vertical net density signifies the probability values indicating how many nets are likely to be routed horizontally/vertically through a G-cell. As shown in the figure, for a net encompassing a bounding box of $W\times H$, $\frac{1}{W}$ and $\frac{1}{H}$ are incorporated into the horizontal and vertical net density values, respectively, across all G-cell grids covered by the net. Notably, a G-cell's net density aggregates the net density values for all nets intersecting the respective G-cell. The division of the density map into horizontal and vertical aspects stems from the consideration that in chip design, horizontal and vertical wires are routed through distinct metal layers to avoid interference between them.

The source/sink location map indicates whether source/sink pins for the target arc exist in a G-cell. This feature results in the generation of n arcs if a net has n fanouts, with these arcs being differentiated by this feature. For instance, in the case of the net illustrated in Fig. 12, it has two fanouts (blue and white) emanating from the source (red). In this scenario, the two arcs share all other features except for the source/sink location. Specifically, if the source/sink location moves from red to blue, it carries a value of 1 at positions (0,0) and (3,1), while if it moves from red to white, it holds a value of 1 at positions (0,0) and (2,1). Lastly, the pin capacitance location map accumulates capacitance values within the G-cells corresponding to the sink pins' positions.

Our proposed CNN prediction model follows the conventional CNN architecture, consisting of both CNN layers and fully connected (FC) layers. The CNN layers are composed of three sequential stages ((5→16→16→16 channels) while the FC layers consist of three stages as well (flatten→100→20→1). During our research, we observed an increase in computation time due to the variability in input features. To address this, we adopted **adaptive average pooling** to transform the input feature's size into $5\times 7\times 7$, enabling parallel batch-wise computation. This adjustment results in a significant reduction in computation time by about 30 times without compromising prediction accuracy in both the training and inference stages. As depicted in Fig. 8, the outcomes of the CNN prediction model can be observed. These predictions are then incorporated into the input feature of the level-2 arc delay/slew prediction model, specifically the pin-to-pin distance. This addition of CNN prediction outcome aids in achieving a more precise prediction on arc delay and slew.

3.5. GNN prediction model for Arc delay and Arc output slew

The architecture of our GNN model for predicting arc delay and slew, corresponding to the right side in Fig. 8, employs the same structure as the GNN model in level-1, utilizing three GNN layers. The input features of the GNN of level-2 are described below:

- **Pre-route arc delay, arc output slew:** Arc delay and Arc output slew used in the arcs are prediction values of commercial EDA tool.
- **Arc input slew:** Arc input slew is slew value at cell outputs obtained from a liberty file.
- **Pin-to-pin distance:** It is Manhattan distance between two pins estimated from the parasitic report.
- **Arc R:** It is accumulated resistance from source to sink.
- **Arc RC(τ):** It signifies an Elmore delay [19] value.
- **Arc Max/Min via layer:** It indicates predictive information regarding which layer to use.
- **Net R, Net C:** Prediction value of level-1 GNN prediction model.

Among the total of 10 input features, net R and net C are obtained from the level-1 GNN prediction model. The pin-to-pin distance feature incorporates the predicted variations from the level-1 CNN prediction model. Furthermore, arc delay and arc output slew used in the arcs are extracted from the pre-route timing report of a commercial EDA tool while the arc input slew is obtained from a liberty file. Arc R is determined by following the path of the net's RC tree from its source to the sink and aggregating the resistances associated with each segment. The parameter τ (RC) signifies an Elmore delay [19] value obtained through the conversion of individual wire segments into a π-model, thereby enabling the computation of the RC delay spanning from the source to the sink. Moreover, in predicting arc delay and arc output slew, the inclusion of details concerning the top-most and bottom-most via layer offers insight into the routing resources prior to actual routing.

3.5.1. Arc based graph

The arc-based graph on the right side of Fig. 8, represents the individual arcs in the circuit. Each node in this graph corresponds to an arc which represents a segment of a net between two connection points. The edges in this graph indicate the relationship between different arcs. We partitioned the entire design layout into n by n grid. For each grid, in the arc-based graph $G_a(V_a, E_a)$, each node $a_i \in V_a$ corresponds to a arc $a_i \in A$, and the node feature of a_i is formulated as $\overrightarrow{a_i}$ which is concatenated vectors of level-2 features and prediction result of level-1. For example, when considering the arc-based graph as depicted in

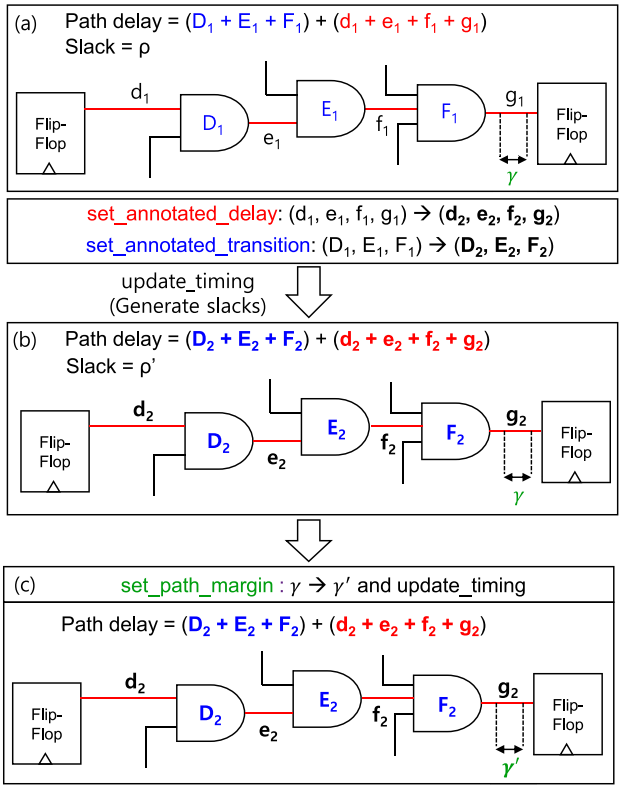

Fig. 14. Illustration of three-step pre-route timing annotation process through EDA tool commands.

Fig. 13, the node $\overrightarrow{a_i}$ can be represented as [Arc delay, Arc input slew, Arc output slew, PD, Arc R, τ, AMaxV, AminV, Net R, Net C] where each element represents the arc delay, input slew of arc, output slew of arc, pin-to-pin distance, resistance of arc, elmore RC of arc, highest via layer of arc, lowest via later of arc and Net R, C from level-1. Similar to the net based graph, arcs are connected through edges based on overlapping bounding boxes. To save run time, instead of comparing all arcs, we restrict the comparison to the arcs within overlapping nets. For an arc-based edge, just as in the net-based graph, each edge $e_{ij} \in E_a$ is a directed edge from $a_j \in V_a$ to $a_i \in V_a$, indicating whether there is an overlap between a_i and a_j. The edge feature $\overrightarrow{e_{ij}}$ of e_{ij} can be expressed as $\overrightarrow{e_{ij}} = (avgx_i, avgy_i, avgx_j, avgy_j, RP_{a_i,a_j})$. Using the same methodology employed in creating embedding vectors in the net-based graph, embedding vectors were also generated in the arc-based graph.

4. Timing optimization linked into EDA tool chain

Our timing optimization framework corresponds to the red boxes and arrows in Fig. 7, in which it makes use of a commercial EDA tool chain. Here, our key idea is to replace the arc delay and arc output slew values predicted by the commercial tool at the pre-route stage with the values produced by our GNN based prediction framework, so that the EDA tool chain optimizes timing more effectively at the post-route stage with more accurate prediction data on the arc delay and arc output slew. While ideally, predicted slew and delay would be directly incorporated into paths, current commercial tools do not support this direct integration. Instead, path margin commands within commercial EDA tools are used to effectively apply these predictions without loss. This approach helps avoid unnecessary large path margin insertions for flip-flops with high-positive worst slacks, enhancing timing optimization. It also prevents outliers, such as extremely inaccurate predictions, from adversely affecting the design quality.

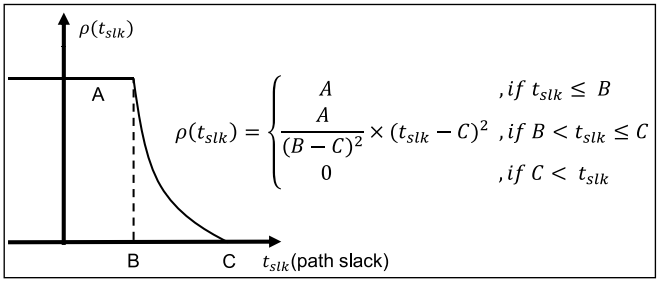

Fig. 15. Margin ratio function $\rho(t_{slk})$ used in our timing optimization framework. The curve indicates that as t_{slk} decreases or increases, a more or less emphasis is placed on optimizing timing on the corresponding circuit path.

Fig. 14 illustrates the pre-route timing annotation process through Synopsys ICC2 tool commands. Note that most commercial tools have a common capability of path delay annotation at the pre-route stage so as to guide the circuit timing optimization process during the routing stage. The pre-route delay and slew annotation are performed in three steps as DTOC [13] does. We elaborate on the process with the example in Fig. 14 in the following.

1. (*Annotating path delay*): The path delay shown in Fig. 14(a) is composed of three gate delays, which are D_1, E_1, and F_1, and four arc delays, which are d_1, e_1, f_1, and g_1). By using an internal tool command like `set_annotated_delay` and `set_annotated_transition` in ICC2, we are able to replace not only the arc delays which are d_1, e_1, f_1, and g_1 with new ones namely d_2, e_2, f_2, and g_2 but also arc output slews which are D_1, E_1, and F_1 with D_2, E_2, and F_2 obtained by our prediction framework, as shown in Fig. 14(a).
2. (*Updating timing*): Since the predicted values of the path delay components are changed in step 1, we need to refresh all path delays in the circuit by using an internal command of updating path timing to obtain the timing slack value on each flip-flop in the circuit. For example, in Fig. 14(b), we use command `update_timing` to update timing, which is available to use in ICC2 tool.
3. (*Annotating path margin*): By using a margin ratio function $\rho(t_{slk})$ like the one shown in Fig. 15, we compute a new path margin which is defined as the difference in path slack before and after annotation on each flip-flop. We annotate the path margins, denoted by γ', to the corresponding flip-flops by using an annotation command such as `set_path_margin` in ICC2, as shown in Fig. 14(c).

Once the annotation process is done at the pre-route stage, we process the subsequent timing optimization flow in the commercial tool chain to optimize the circuit path timing by exploiting the more accurate data, which has been set by the path margin γ', as illustrated in Fig. 14(c).

The key element that affects the timing quality is how the margin ratio function can be effectively utilized. Fig. 15 shows the margin ratio function we use. The function curve changes as the values of the parameters A, B, and C change. Unlike DTOC [13] which fixes the values A, B, and C in Fig. 15, independently of circuits to be optimized, the shape of our margin ratio function in Fig. 15 varies according to the circuit characteristics which include clock, utilization, gate count, net count, chip area, flip-flop count, and the quantities of buffers and inverters.

Through empirical analysis in the course of setting the various parameter values of A, B, and C in the margin ratio function, we observed that the timing optimization results for each circuit exhibit significant variations. Based on this observation, accurately identifying the appropriate values of A, B, and C, in accordance with circuit

Table 2
Benchmark circuit information.

Test bench	Training dataset							Test dataset						
	Clk. (ns)	Util.	#Gates	#Nets	Chip area	#FF	#BUF&INV	Clk. (ns)	Util.	#Gates	#Nets	Chip area	#FF	#BUF&INV
ecg	0.5	70	115,425	116,338	59 754.775	14,036	28,341	0.4	60	114,454	115,367	69 831.229	14,036	27,335
ethernet	0.5	60	46,306	46,306	36 950.016	10,543	14,778	0.5	70	46,091	46,196	317 476.687	10,543	14,478
jpeg	0.7	70	243,458	265,187	160 768.918	37,521	78,551	0.7	60	266,319	288,048	187 620.655	37,521	84,969
ldpc	0.5	80	44,173	47,620	19 341.017	2048	7479	0.6	70	44,453	47,900	22 046.245	2048	7759
nova	1.1	60	159,279	161,858	118 402.056	29,098	48,176	1.0	70	158,945	161,519	101 296.964	29,098	47,800
tate	0.6	70	243,360	244,119	128 886.620	31,420	55,555	0.45	65	244,297	245,057	138 908.221	31,420	55,125
vga	0.65	70	72,821	72,910	50,348.163	17,067	24,728	0.65	60	73,120	73,209	58 664.632	17,067	24,989
wb_conmax	0.4	80	22,449	23,581	8212.709	818	5258	0.4	55	23,068	24,200	11 914.15	818	11,225
aes_128	–	–	–	–	–	–	–	0.7	50	105,361	105,621	52 071.580	10,688	23,540
des3	–	–	–	–	–	–	–	0.7	50	52,776	53,012	30 170.284	8808	14,896

characteristics, tends to produce high-quality implementations when the corresponding margin ratio functions are applied to the commercial EDA tools.

As a result, XGBoost, known for its effectiveness in machine learning regression tasks, was selected as the appropriate approach to accurately driving the prediction. Separate XGBoost models are constructed for A, B, and C. For each A, B, and C variation, annotation of prediction delay and slew was conducted, with results derived for 100 test benches each. Based on these results, the top 10 for each test benches were selected to obtain the label values for A, B, and C. Following this, XGBoost was trained using these labels, and subsequently used to predict A, B, and C. In the training set, seven circuit characteristics — clock, utilization, gate count, net count, chip area, flip-flop count, buffer, and inverter count — are selected as input features.

As demonstrated in Table 2, the chip area and the number of buffers and inverters change according to clock and utilization, hence they have been selected as features to illustrate differences within the same test bench. Gate count, net count, chip area, flip-flop count, buffer, and inverter count also vary not only with clk and utilization but according to the function of each test bench, and significantly influence the routing paths and arcs. Therefore, these were chosen as representative features for the test benches.

By annotating delay and slew, the values of A, B, and C that lead to the most effective timing optimization are determined and labeled for the training process. Subsequently, for the test set, our XGBoost models of A, B, and C are utilized to predict their respective values. These predicted values are then applied to the margin ratio function of the individual circuits.

5. Experimental results

5.1. Experimental setup

We implemented our prediction model in Pytorch and train it with Tesla V100-DGXS-32GBGPU. In addition, we implemented our timing optimization flow in C++ and Python. We generated TCL script for P&R tool to annotate the predicted delays and slews after the stage of clock tree synthesis (i.e., `clock_opt` in ICC2), updated timing information, and assigned path margins. Unlike to DTOC [13], in which the parameter values of A, B, and C were constant for all circuits, we tuned the values according to the circuit characteristics which included the gate count, net count, chip area, flip-flop count, and the quantities of buffers and inverters.

To generate our training datasets, we implemented the open-source benchmark circuits in [25]. We used Nangate 15 nm open cell library in [26,27] for the implementation. Table 2 shows the information of benchmark circuits used in our experiments which includes circuit characteristics. We created training and test datasets separately, each of which consists of eight distinct sets. Our prediction model is trained with two methods which are called ‘seen’, and ‘unseen’ respectively with average time 5 h. The ‘seen’ test includes arcs in all 8 training designs, while the ‘unseen’ test has arcs from 7 training designs except for the target design we want to test. For instance, when conducting a ‘seen’ test for the ECG design, training is carried out using the training datasets of all eight designs listed in Table 2, followed by testing using the test dataset of the ECG design. Conversely, during an ‘unseen’ test for the ECG design, the training employs the training datasets from the other seven designs, excluding ECG, and proceed test with test dataset of the ECG design. Additionally, as a form of unseen test, the test benches aes_128 and des3, which had not been used previously in training or testing, were employed.

5.2. Assessing prediction accuracy

We assess the delay and slew qualities predicted at the `clock_opt` pre-route stage by the ICC2 commercial tool, DTOC in [13], and our prediction framework by comparing them with the actual implementation quality reported at the post-route stage by the ICC2 tool. We use the mean square error (MSE) as an assessment metric.

5.2.1. Comparing prediction accuracy

Table 3 shows a comparison of MSE values on the three components of level-1 which are net R, net C, arc length. For most of the test cases, our prediction framework produces the smaller than that by the ICC2 tool and DTOC for all three components. The prominent accuracy enhancement on net R and net C prediction of seen test (i.e., 62.66% and 29.26% over the MSE of ICC2) implies that our GNN model to predict net R and net C is very effective. Furthermore, a considerable improvement in accuracy was also observed in the case of unseen tests. (i.e., 59.21% and 26.76% over the MSE of ICC2). On the other hand, the arc length prediction which uses a CNN model, focusing on net congestion related input features as DTCO does, produces the same and similar level of prediction accuracy as that of DTOC in the seen and unseen test.

By examining the comparison of arc slew prediction shown in Table 4, integrating the prediction values of net R, net C, and arc length taken from *level-1* as input features into our GNN model for predicting arc slew at *level-2* makes significant improvements in accuracy in both seen (i.e., 36.97% over the MSE of ICC2 and 35.76% over the MSE of DTOC) and unseen tests (i.e., 29.79% over the MSE of ICC2 and 28.58% over the MSE of DTOC).

At level-2, both slew and delay are predicted. The output slew predicted at level-2 is used as an input feature for predicting delay at the same level, allowing for more precise delay predictions. This approach effectively improved delay prediction accuracy: in seen tests, there was a 42.54% improvement compared to ICC2 and up to a 31% improvement over other previous studies. In unseen tests, there was a 35.96% improvement compared to ICC2 and up to a 25% improvement over previous research.

Table 3
Comparison of level-1 prediction accuracy, in terms of MSE (mean square error), of Synopsys ICC2 timing reports, DTOC [13], and our GNN based prediction framework.

Design	Net R. MSE.				Net C. MSE.				Arc length. MSE.			
	ICC2	DTOC [13] (impr.)	Ours (impr.)		ICC2	DTOC [13] (impr.)	Ours (impr.)		ICC2	DTOC [13] (impr.)	Ours (impr.)	
			Seen	Unseen			Seen	Unseen			Seen	Unseen
ecg	0.1151	0.0544 (52.69%)	0.0329 (71.41%)	0.0338 (70.61%)	0.5241	0.3367 (35.75%)	0.3629 (30.76%)	0.3691 (29.59%)	1.0153	0.9692 (4.54%)	0.9692 (4.54%)	1.014 (0.13%)
ethernet	0.2570	0.1132 (55.94%)	0.0581 (72.79%)	0.0612 (76.20%)	0.9982	0.6435 (35.53%)	0.5477 (46.16%)	0.6243 (37.46%)	10.9878	10.8461 (1.29%)	10.8461 (1.29%)	10.9302 (0.52%)
jpeg	0.1779	0.1229 (30.90%)	0.0837 (52.91%)	0.1294 (27.26%)	0.3376	0.2973 (11.92%)	0.2936 (13.04%)	0.2707 (19.81%)	1.9307	1.8617 (3.57%)	1.8617 (3.57%)	1.7771 (7.95%)
ldpc	2.1282	0.5895 (72.29%)	0.1913 (91.01%)	0.2633 (87.63%)	1.5439	0.7611 (50.69%)	0.8798 (43.01%)	0.9354 (39.43%)	37.1083	36.6130 (1.33%)	36.6130 (1.33%)	36.6479 (1.24%)
nova	0.5618	0.3078 (45.21%)	0.2582 (54.03%)	0.2748 (51.08%)	1.3533	1.4178 (−4.76%)	1.1217 (17.12%)	1.1891 (12.13%)	11.7010	11.5304 (1.46%)	11.5304 (1.46%)	11.5498 (1.29%)
tate	0.4985	0.4993 (−1.81%)	0.2370 (51.67%)	0.2362 (51.83%)	1.1564	0.9361 (19.04%)	0.8993 (22.23%)	0.9521 (17.66%)	2.7261	2.6638 (2.28%)	2.6638 (2.28%)	1.1758 (3.57%)
vga	1.6033	1.2847 (19.87%)	0.7722 (51.83%)	0.8011 (50.04%)	2.8972	2.2256 (23.17%)	2.1413 (26.08%)	2.1111 (27.13%)	9.7248	9.5813 (1.48%)	9.5813 (1.48%)	9.6312 (0.96%)
wb_conmax	0.1360	0.0575 (57.72%)	0.0608 (55.29%)	0.0557 (59.06%)	0.3761	0.2410 (35.91%)	0.2418 (35.70%)	0.2599 (30.88%)	3.9083	3.8364 (1.84%)	3.8364 (1.84%)	3.7965 (4.32%)
avg		41.60%	**62.66%**	**59.21%**		25.90%	**29.26%**	**26.76%**		2.22%	2.22%	2.49%

Table 4
Comparison of slew prediction accuracy, in terms of MSE (mean square error), of Synopsys ICC2 timing reports, DTOC [13] and our GNN based prediction framework.

Design	Arc slew. MSE.			
	ICC2	DTOC (impr.)	Ours (impr.)	
			Seen	Unseen
ecg	22.8086	21.3032 (6.60%)	17.8420 (21.78%)	17.7681 (22.10%)
ethernet	172.8331	153.3933 (11.25%)	119.1204 (31.08%)	113.3438 (34.42%)
jpeg	363.8876	662.8265 (−82.15%)	298.3953 (18.00%)	261.4052 (28.16%)
ldpc	1547.9711	789.3869 (49.87%)	593.2960 (61.67%)	854.4229 (44.80%)
nova	515.2590	563.6834 (−9.39%)	262.6059 (49.03%)	347.1886 (32.62%)
tate	261.6313	238.5332 (8.83%)	154.5344 (40.93%)	182.9015 (30.09%)
vga	2271.5701	2220.6072 (2.24%)	1281.6654 (43.58%)	1424.8229 (37.28%)
wb_conmax	61.4186	47.6587 (22.40%)	43.1680 (29.72%)	43.1209 (29.79%)
avg		1.21%	**36.97%**	**29.79%**

5.2.2. Analysis on prediction quality

As revealed from the MSE comparison in Table 3, DTOC [13] which is a fully CNN based model tends to make prediction accuracy improvements over the prediction by a commercial tool. However, there some test cases where DTOC performs worse than the commercial tool. For example, for circuit tate, the net R prediction by DTOC is less accurate than ICC2 tool. This is mainly because of the lack of GNN based model that represents the physical proximity and overlapping relation among the nets. On the other hand, our GNN based outperforms the ANN based model in DTOC in net R prediction. However, contrary to the net R prediction, the prediction accuracy of net C prediction made by our GNN based model is not so much impressive, i.e. 3.36% more MSE improvements than DTOC. However, predicting net C involves additional considerations This is mainly because of the lack of some features that both of our GNN based and DTOC's CNN based models may need.

Finally, for the arc slew prediction, our GNN based model works very well, i.e., 36.97% and 35.76% less MSE than the MSE of ICC2 and DTOC, respectively. Additionally, for the arc delay prediction, our GNN based model works very well, i.e., 42.54% and 32.49% less MSE than the MSE of ICC2 and DTOC. In Tables 4 and 5, our GNN model demonstrates better results compared to DTOC. This indicates that our GAT based GNN model effectively and accurately represents the diverse relationships between arcs in circuit. Furthermore, our GNN model improves the accuracy of delay and slew compared to the GNN-do model, which predicts delay and slew without the level-1 net R, net C. It demonstrated the effectiveness of hierarchical model.

Through comparisons between GNN-rc, a model derived from our proposed study excluding CNN, and our GNN model, we were able to assess the impact of CNN on the prediction aspects. CNN represents the congestion in circuits, which had been limited to a single numerical value in the past. However, by depicting congestion in 2D, CNN has refined its representation, aiding in the prediction of detour nets. Experiments confirmed that this approach not only helped in predicting detour nets but also enhanced the overall prediction results. Moreover, our final model, which incorporates newly predicted output slew, showed about a 3% improvement over the GNN-dd model that predicted delay without updating the output slew. This confirms that more precise prediction of output slew enhances the accuracy of delay prediction. Additionally, this research demonstrated an improvement of 35.96% and 24.53% in delay prediction over previous GNN-based studies, TIG [14] and WDGCN [28] respectively. This suggests difficulties in predicting detour routing nets when neither routing congestion is considered, nor when graphs are constructed based solely on gates. Our GNN model demonstrates better results compared to DTOC [13], GCN [21], GCNII [29], which treat the relationships between surrounding arcs with equal importance. This indicates that our GAT based GNN model effectively and accurately represents the diverse relationships between arcs in circuit.

Such a high improvement of our final GNN model is attributed to two factors: (1) using GAT based GNN model in *level-2* for representing neighboring arcs and their connectivity, surpassing simple numerical comparisons and effectively representing the circuits, and (2) using the prediction values of net R, net C, and arc length produced in *level-1* with much improved accuracy as additional input features of the

Table 5
Comparison of level-2 prediction accuracy, in terms of MSE (mean square error), of Synopsys ICC2 timing reports, DTOC [13], previous works [14,21,28,29], and our GNN based prediction framework.

Design	Arc delay. MSE.											
	ICC2	DTOC [13] (impr.)	TIG [14] (impr.)	WDGCN [28] (impr.)	GCN [21] (impr.)	GCNII [29] (impr.)	GNN-do (impr.)	GNN-rc (impr.)	GNN-dd (impr.)	Ours (impr.)		Time (s)
										Seen	Unseen	
ecg	3.4592	3.0094 (12.99%)	4.2413 (−22.61%)	3.8473 (−11.22%)	2.8866 (16.55%)	4.0843 (−18.07%)	3.6984 (−6.91%)	3.1894 (7.80%)	2.7390 (20.82%)	2.5326 (26.78%)	2.7622 (20.14%)	2.4146
ethernet	24.8932	17.0944 (31.32%)	25.0655 (−0.69%)	18.9138 (24.02%)	19.6576 (21.03%)	25.0242 (−0.52%)	21.9588 (11.78%)	16.9642 (31.85%)	15.0335 (39.60%)	15.5605 (37.49%)	16.0479 (35.53%)	1.8284
jpeg	29.8783	41.1782 (−37.82%)	25.6544 (14.20%)	23.9730 (19.76%)	22.0529 (26.24%)	19.5092 (34.70%)	18.1844 (39.14%)	18.6441 (37.59%)	17.6764 (40.84%)	17.3030 (42.08%)	22.5421 (24.55%)	1.6757
ldpc	194.1034	125.1312 (35.53%)	179.6706 (7.43%)	109.7113 (43.48%)	90.8156 (53.21%)	90.7869 (53.23%)	105.5627 (45.61%)	95.9760 (50.56%)	98.5847 (49.21%)	85.8095 (55.79%)	99.1247 (48.93%)	5.7106
nova	85.6362	75.5739 (11.75%)	61.2691 (28.45%)	63.7521 (15.48%)	44.4803 (41.03%)	52.4951 (30.40%)	41.7903 (44.59%)	36.7502 (51.28%)	54.3057 (36.59%)	36.9271 (51.05%)	45.5541 (46.80%)	3.1627
tate	37.0706	37.7212 (−1.75%)	38.6588 (−4.28%)	30.3152 (18.22%)	31.8460 (14.09%)	28.8098 (22.28%)	34.8737 (5.92%)	25.4445 (31.36%)	23.7398 (35.96%)	22.1748 (40.18%)	27.5730 (25.62%)	2.0620
vga	333.5998	380.2917 (−13.99%)	257.5434 (22.79%)	283.4435 (15.03%)	182.3483 (45.34%)	208.2724 (37.57%)	276.3732 (17.15%)	208.5932 (37.47%)	182.2476 (45.37%)	164.6769 (50.64%)	198.5644 (40.47%)	3.9109
wb_conmax	8.7270	6.6568 (23.72%)	7.4074 (15.12%)	7.4523 (14.60%)	7.1698 (17.84%)	8.6309 (1.10%)	7.5389 (13.61%)	7.0640 (19.06%)	6.4493 (26.10%)	6.5633 (24.79%)	6.7553 (22.59%)	2.3190
aes_128	1.2013	0.8593 (28.47%)	0.9043 (24.72%)	0.6876 (42.76%)	0.6760 (43.72%)	0.6872 (42.80%)	0.6588 (45.16%)	0.6810 (43.31%)	0.3395 (71.74%)	0.3857 (67.30%)		1.8663
des3	0.2077	0.1738 (16.32%)	0.2477 (−19.25%)	0.2118 (−1.95%)	0.2425 (−16.76%)	0.4112 (−97.98%)	0.2729 (−31.34%)	0.2058 (0.96%)	0.1526 (26.53%)	0.1480 (27.73%)		0.7504
avg		10.05%	6.58%	18.01%	26.22%	10.55%	18.47%	34.12%	**39.28%**	**42.54%**	**35.96%**	

Table 6
Comparison of the timing prediction of different layer of GNN timing optimization framework.

Design	Arc delay MSE.						
	ICC2	Layer2 (impr.)	Ours (impr.)	Layer4 (impr.)	Layer5 (impr.)	Layer6 (impr.)	Layer7 (impr.)
ecg	3.4591	2.7365 (20.88%)	2.5326 (26.78%)	2.9656 (14.26%)	3.0781 (11.01%)	2.7156 (21.49%)	2.9559 (14.54%)
ethernet	24.8932	16.9876 (31.75%)	17.3030 (37.49%)	19.2410 (22.70%)	19.2115 (22.82%)	15.8232 (36.43%)	20.3775 (18.14%)
jpeg	29.8983	42.5160 (17.77%)	17.3030 (42.08%)	19.4674 (34.84%)	22.7159 (23.97%)	24.1831 (19.06%)	18.9616 (36.53%)
ldpc	194.1098	90.1369 (53.56%)	85.8095 (55.79%)	83.7371 (56.86%)	105.3015 (45.75%)	107.3217 (44.71%)	93.6221 (51.76%)
nova	75.4329	31.7243 (57.94%)	36.9271 (51.05%)	33.8404 (55.13%)	36.3228 (51.84%)	37.5750 (50.18%)	41.5519 (34.49%)
tate	37.0706	24.7215 (33.31%)	22.1748 (40.18%)	24.7152 (33.32%)	25.3597 (31.59%)	23.4267 (36.80%)	24.2833 (34.49%)
vga	333.5998	181.0986 (45.71%)	164.6769 (50.64%)	207.3729 (37.83%)	212.2379 (36.37%)	202.9994 (39.14%)	199.1474 (40.30%)
wb_conmax	8.727	6.7248 (22.94%)	6.5633 (24.79%)	6.4717 (25.84%)	6.9905 (19.89%)	6.709 (23.12%)	7.0121 (34.03%)
avg		35.48%	**41.10%**	33.15%	29.36%	33.11%	31.07%

GAT based GNN model in *level-2*. Furthermore, as evidenced by the results of both the seen and unseen tests, our framework demonstrated a significant degree of accuracy in predicting delays and slews not only in the seen test but also in the unseen ones. Additionally, the model performed impressively in predicting delays for entirely different test benches, aes_128 and des3, which were not used at all during the training process. This suggests that our framework can be leveraged to anticipate designs that were not encountered during the training phase, signifying a high level of generality.

Table 6 shows the delay prediction results based on the number of layers in the GNN. Our final GNN model with three layers demonstrates the best performance. This indicates that increasing the number of layers can lead to an over-smoothing issue, where embeddings become too similar overall, hindering precise predictions.

5.3. Assessing timing optimization

Table 7 shows a comparison of the values of worst negative slack (WNS), total negative slack (TNS), and the number of timing violation paths (#VP) on the implementation of circuits produced by Synopsys ICC2, DTOC in [13], and our timing prediction and optimization framework. In summary, our framework consumed an average of 1.33x more time which includes inference time, annotation time and routing time, compared to ICC2 routing time, but resulted in a further reduced WNS by 77%, TNS by 77%, and #VP by 64% on

Table 7
Comparison of the timing optimization quality, in terms of WNS (worst negative slack), TNS (total negative slack), and #VP (timing violation path count), of the circuit implementations produced by Synopsys ICC2, DTOC [13], and our timing optimization framework.

Test bench	ICC2				DTOC					Ours				
	WNS	TNS	#VP	Time (s)	Delay/Slew prediction Impr. A/B/C	WNS (impr.)	TNS (impr.)	#VP (impr.)	Time (s) (Inc.)	Delay/Slew prediction Impr. A/B/C	WNS (impr.)	TNS (impr.)	#VP (impr.)	Time (s) (Inc.)
ecg	−29.44	−837.38	66	927	12.99%/6.60% 1.0/0.1/0.5	0.84 (102.85%)	0 (100.00%)	0 (100.00%)	1231.34 (1.32x)	26.78%/21.78% 1.0/0.1/0.85	23.45 (179.65%)	0 (100.00%)	0 (100.00%)	1129.90 (1.22x)
ethernet	−78.29	−1581.14	49	460	31.32%/11.25% 1.0/0.1/0.5	−44.09 (43.68%)	−1091.24 (30.98%)	62 (−26.53%)	669.57 (1.46x)	37.49%/31.08% 1.0/0.3/0.5	−5.40 (93.10%)	−21.45 (98.64%)	6 (87.76%)	602.32 (1.31x)
jpeg	−72.08	−1206.85	56	3167	−32.82%/−82.15% 1.0/0.1/0.5	−61.93 (14.08%)	−655.23 (45.71%)	36 (35.71%)	4082.31 (1.29x)	42.08%/18.00% 1.0/0.25/0.8	−42.21 (41.44%)	−398.45 (66.98%)	19 (66.07%)	3693.50 (1.17x)
ldpc	−128.78	−977.67	19	406	35.53%/49.87% 1.0/0.1/0.5	−80.7 (37.33%)	−885.42 (9.44%)	23 (−21.05%)	612.84 (1.51x)	55.79%/61.67% 1.0/0.15/0.7	−54.11 (57.98%)	−498.84 (48.98%)	14 (26.32%)	598.15 (1.47x)
nova	−75.27	−6601.72	179	1724	11.75%/−9.39% 1.0/0.1/0.5	−21.39 (71.58%)	−251.76 (96.19%)	17 (90.50%)	2423.56 (1.41x)	51.05%/49.03% 1.0/0.24/0.6	−30.10 (60.01%)	−2098.60 (68.21%)	118 (34.08%)	2241.77 (1.30x)
tate	−155.14	−39 768.02	916	2050	−1.75%/8.83% 1.0/0.1/0.5	−145.49 (6.22%)	−33 312.49 (16.23%)	1283 (−40.07%)	3011.86 (1.47x)	40.18%/40.93% 1.0/0.25/0.4	−95.37 (38.53%)	−10 442.56 (73.74%)	351 (61.68%)	2846.99 (1.39x)
vga	−62.64	−2552.08	135	804	−13.99%/2.24% 1.0/0.1/0.5	−5.71 (90.88%)	−11.7 (99.54%)	3 (97.78%)	1511.47 (1.88x)	50.64%/43.58% 1.0/0.1/0.45	11.21 (117.90%)	0 (100.00%)	0 (100.00%)	1027.84 (1.27x)
wb_conmax	−72.47	−2298.88	108	233	23.72%/22.40% 1.0/0.1/0.5	−61.73 (14.82%)	−873.61 (62.00%)	73 (32.41%)	312.27 (1.34x)	24.79%/29.72% 1.0/0.15/0.8	−52.65 (27.35%)	−924.15 (59.80%)	65 (39.81%)	343.39 (1.47x)
avg					7.71%/1.21%	47.68%	57.51%	33.59%	1.46x	41.10%/36.97%	**77.00%**	**77.04%**	**64.46%**	1.33x

average over that produced by ICC2 tool. Moreover, in comparison to the implementations produced by DTOC [13], ours produces implementations with 29% less WNS, 19% less TNS, and 30% fewer #VP on average while using less average time. Specifically, for circuit JPEG, the high accuracy improvement (i.e., 42.08%/18.00%) in arc delay and slew prediction has made a very positive effect on timing optimization. On the other hand, for circuit WB_CONMAX, though our prediction accuracy (i.e., 24.79%/29.72%) is not so high over that of DTOC (i.e., 23.72%/22.40%), tuning the parameter values of A, B, and C with XGBoost in the margin ratio function in Fig. 15 effectively optimizes the circuit timing, producing implementations with 49.42% less WNS, 67.66% less TNS, and 114.29% fewer #VP over that by DTOC.

6. Conclusion

This work proposed a novel timing prediction and optimization framework that included a set of GNN based pre-route prediction models and an integration of the prediction models into a commercial tool chain for optimizing circuit timing. Specifically, to achieve timing prediction, we analyzed and identified a set of core factors influencing the delay and slew on net arcs, and developed a GNN based timing prediction framework. Through GNN based models, we were able to consider not only the features directly affecting the delay and slew on arcs but also incorporate the proximity relationship among the arcs. Furthermore, we established a hierarchical two-stage prediction model where *level-1* predicted net R, net C, and arc length, whose values were then fed to *level-2* to predict arc delay and arc slew. Through experiments, it was shown that our prediction framework made an accuracy improvement of 35.96~42.54% on average in arc delay prediction and 29.79~36.97% in arc slew prediction over that achieved by the conventional commercial tool and improvement of upto 35.96% and upto 35.76% over that by state of the art framework [13,14,29]. Furthermore, we developed an advanced methodology to effectively incorporate the predicted arc delay and slew into a commercial tool chain for timing optimization. In summary, our timing optimization method was able to further reduce the worst negative slack, total negative slack, and the number of timing violation paths by 77.00%, 77.04%, and 64.46% in comparison with that produced by a commercial tool, and by 29.32%, 19.53%, and 30.87% in comparison with that by DTOC [13], respectively.

Declaration of competing interest

The authors declare that the paper is original and has not been published or submitted to any of workshops, conferences and journals.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was supported by Samsung Electronics Company, Ltd. (IO201216-08205-01, and IO221227-04376-01), Samsung Advanced Institute of Technology (IO230223-05124-01), National Research Foundation grant by Korea Government (2021-R1A2C2008864), Institute of Information and communications Technology Planning and Evaluation (IITP) grant by Korea government (2021-0-00754), Software Systems for AI Semiconductor Design and Artificial Intelligence Semiconductor Support Program to nurture the best talents (IITP-2023-RS-2023-00256081) grant by Korea government, National R&D Program through NRF by Ministry of Science and ICT (2020M3H2A1078119), and BK21 Four Program of Education and Research Program for Future ICT Pioneers, Seoul National University. The EDA tool was supported by IC Design Education.

References

[1] T. Huynh-Bao, J. Ryckaert, Z. Tőkei, A. Mercha, D. Verkest, A.V.-Y. Thean, P. Wambacq, Statistical timing analysis considering device and interconnect variability for beol requirements in the 5-nm node and beyond, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 25 (5) (2017) 1669–1680.

[2] Z. Xie, H. Ren, B. Khailany, Y. Sheng, S. Santosh, J. Hu, Y. Chen, Powernet: Transferable dynamic ir drop estimation via maximum convolutional neural network, in: IEEE/ACM Asia and South Pacific Design Automation Conference, ASP-DAC, 2020, pp. 13–18.

[3] J. Chen, J. Kuang, G. Zhao, D.J.-H. Huang, E.F. Young, Pros: A plug-in for routability optimization applied in the state-of-the-art commercial eda tool using deep learning, in: IEEE/ACM International Conference on Computer-Aided Design, ICCAD, 2020, pp. 1–8.

[4] K. Baek, H. Park, S. Kim, K. Choi, T. Kim, Pin accessibility and routing congestion aware drc hotspot prediction using graph neural network and u-net, in: IEEE/ACM International Conference on Computer-Aided Design, ICCAD, 2022, pp. 1–9.

[5] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, J. Hu, Routenet: Routability prediction for mixed-size designs using convolutional neural network, in: IEEE/ACM International Conference on Computer-Aided Design, ICCAD, 2018, pp. 1–8.

[6] A.B. Kahng, S. Kang, H. Lee, S. Nath, J. Wadhwani, Learning-based approximation of interconnect delay and slew in signoff timing tools, in: IEEE/ACM International Workshop on System Level Interconnect Prediction, SLIP, 2013, pp. 1–8.

[7] S.-S. Han, A.B. Kahng, S. Nath, A.S. Vydyanathan, A deep learning methodology to proliferate golden signoff timing, in: IEEE/ACM Design, Automation & Test in Europe Conference & Exhibition, DATE, 2014, pp. 1–6.

[8] A.B. Kahng, M. Luo, S. Nath, Si for free: machine learning of interconnect coupling delay and transition effects, in: IEEE/ACM International Workshop on System Level Interconnect Prediction, SLIP, 2015, pp. 1–8.

[9] J. Liu, S. Park, J. Seomun, H.-O. Kim, J.Y. Choi, A machine learning based p & r flow to enhance pre-route vs post-route r/c correlation, in: IEEE/ACM Design Automation Conference, DAC, 2020.
[10] E.C. Barboza, N. Shukla, Y. Chen, J. Hu, Machine learning-based pre-routing timing prediction with reduced pessimism, in: IEEE/ACM Design Automation Conference, DAC, 2019, pp. 1–6.
[11] R. Liang, Z. Xie, J. Jung, V. Chauha, Y. Chen, J. Hu, H. Xiang, G.-J. Nam, Routing-free crosstalk prediction, in: IEEE/ACM International Conference on Computer-Aided Design, ICCAD, 2020, pp. 1–9.
[12] H. Chang, S.S. Sapatnekar, Statistical timing analysis considering spatial correlations using a single pert-like traversal, in: IEEE/ACM International Conference on Computer-Aided Design, ICCAD, 2003, pp. 621–625.
[13] K. Chang, J. Ahn, H. Park, K.-M. Choi, T. Kim, Dtoc: integrating deep-learning driven timing optimization into the state-of-the-art commercial eda tool, in: IEEE/ACM Design, Automation & Test in Europe Conference & Exhibition, DATE, 2023, pp. 1–6.
[14] Z. Guo, M. Liu, J. Gu, S. Zhang, D.Z. Pan, Y. Lin, A timing engine inspired graph neural network model for pre-routing slack prediction, in: IEEE/ACM Design Automation Conference, DAC, 2022, pp. 1207–1212.
[15] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, L. Shi, Graph-learning-driven path-based timing analysis results predictor from graph-based timing analysis, in: IEEE/ACM Asia and South Pacific Design Automation Conference, ASP-DAC, 2023, pp. 547–552.
[16] Y. Ye, T. Chen, Y. Gao, B. Yu, L. Shi, Fast and accurate wire timing estimation based on graph learning, in: IEEE/ACM Design, Automation & Test in Europe Conference & Exhibition, DATE, 2023, pp. 1–6.
[17] Y.-C. Lu, S. Nath, V. Khandelwal, S.K. Lim, Doomed run prediction in physical design by exploiting sequential flow and graph learning, in: IEEE/ACM International Conference on Computer Aided Design, ICCAD, 2021, pp. 1–9.
[18] Q. Song, X. Cheng, P. Cao, Critical paths prediction under multiple corners based on bilstm network, in: IEEE/ACM Design Automation Conference, DAC, 2023.
[19] W.C. Elmore, The transient response of damped linear networks with particular regard to wideband amplifiers, J. Appl. Phys. 19 (1) (1948) 55–63.
[20] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Netw. 20 (1) (2008) 61–80.
[21] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, 2016, arXiv preprint arXiv:1609.02907.
[22] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, 2017, arXiv preprint arXiv:1710.10903.
[23] S. Albawi, T.A. Mohammed, S. Al-Zawi, Understanding of a convolutional neural network, in: IEEE International Conference on Engineering and Technology, ICET, 2017, pp. 1–6.
[24] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: ACM International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794.
[25] Oliscience, Opencores, 1999, [Online]. Available: https://opencores.org.
[26] Nangate freepdk15 open cell library. [Online]. Available: http://www.nangate.com/?pageid=2328.
[27] K. Bhanushali, W.R. Davis, Freepdk15: An open-source predictive process design kit for 15 nm finfet technology, in: ACM International Symposium on Physical Design, ISPD, 2015.
[28] P. Shrestha, S. Phatharodom, I. Savidis, Graph representation learning for gate arrival time prediction, in: Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD, 2022, pp. 127–133.
[29] M. Chen, Z. Wei, Z. Huang, B. Ding, Y. Li, Simple and deep graph convolutional networks, in: International Conference on Machine Learning, PMLR, 2020, pp. 1725–1735.