A Timing Engine Inspired Graph Neural Network Model for Pre-Routing Slack Prediction

Zizheng Guo[1,2†], Mingjie Liu[3†], Jiaqi Gu[3], Shuhan Zhang[3], David Z. Pan[3], Yibo Lin[2*]
[1]School of Computer Science, Peking University [2]School of Integrated Circuits, Peking University
[3]Department of Electrical and Computer Engineering, The University of Texas at Austin
{gzz,yibolin}@pku.edu.cn, {jay_liu,jqgu,shuhan.zhang,dpan}@utexas.edu

ABSTRACT

Fast and accurate pre-routing timing prediction is essential for timing-driven placement since repetitive routing and static timing analysis (STA) iterations are expensive and unacceptable. Prior work on timing prediction aims at estimating net delay and slew, lacking the ability to model global timing metrics. In this work, we present a timing engine inspired graph neural network (GNN) to predict arrival time and slack at timing endpoints. We further leverage edge delays as local auxiliary tasks to facilitate model training with increased model performance. Experimental results on real-world open-source designs demonstrate improved model accuracy and explainability when compared with vanilla deep GNN models.

1 INTRODUCTION

Fast and accurate timing prediction is essential for timing-driven placement since accurate timing information is available only after routing. Placement significantly affects the quality of downstream routing tasks, while it consumes only a fraction of time in the back-end design flow stage when compared with routing. Since repetitive routing and static timing analysis (STA) are expensive and unacceptable during the placement stage, state-of-the-art analytical placers [17, 19] optimize the half-perimeter wirelength as the surrogate of design quality. Although advancements have been made to include further routing congestion and cell density [10, 11] in the optimization objectives, directly targeting routed wirelength, optimization for timing-oriented metrics in large-scale designs remains challenging without a fast and accurate pre-routing timing prediction model.

Early works proposed a variety of heuristics and wire load models to form methods of approximation to consider routing-induced parasitic effects prior to routing [4, 24]. Recent works have demonstrated great success in applying machine learning techniques to estimate the effects of downstream tasks with higher precision, enabling design pruning and optimization in the early upstream design stages. In [12], the authors propose a deep learning methodology to correct divergence and improve the correlation of timing results from different timing engines. The work of [6] uses support vector machines to predict timing failures of embedded memory during the initial floorplan design, significantly reducing the design cycle. Barboza et al. [5] develop a random forest model with extracted placement features to predict routed net delay and slew, where the overall circuit timing is then obtained through PERT traversals [7]. Liang et al. [16] propose a machine learning-based routing-free crosstalk prediction framework, leveraging neighboring net information and graph-based learning methods. Similarly, in analog design, recent works have leveraged deep learning models to predict parasitic layout effects and post-layout performance for device sizing [18] and placement optimization [15].

Recent advances in graph neural networks (GNNs) have demonstrated superior efficacy in learning graph structures and mining graph information [13]. Since circuit netlists can be naturally represented as graphs, GNNs have gained increased popularity in the electronic design automation (EDA) community [16, 18, 20, 21, 23, 25, 27]. However, the performance of GNNs is known to gradually decrease with an increasing number of layers due to *over-smoothing*, where repeated graph convolutions make node embeddings indistinguishable [8, 14]. Thus, prior works in EDA leveraging GNNs can only retrieve local features in learned node embeddings, and graph layers are seldom stacked beyond more than 4 layers. This makes learning node embedding with global information extremely challenging since the limited number of stacked graph convolution layers severely restricts the receptive field of conventional GNNs.

This work presents a timing engine inspired graph neural network model to predict the arrival time and slack value at timing endpoints. Compared with prior work [5] which only leverages local net features in predicting the net delay and slew, our GNN model learns global timing metrics in an end-to-end fashion without additional feature engineering and invoking STA tools. We further explore recent methods in stacking deep graph convolution (GCN) layers [8] and call attention to the community that deep GNN architectures suitable for EDA problems are critical, challenging, yet an underexplored issue. Our method is evaluated on real-world open-source designs [1] with timing data generated from open-source EDA tools [3]. We highlight our contributions:

- To the best of our knowledge, we are the first to present an end-to-end graph learning framework for predicting pre-routing arrival time and slack values at timing endpoints without invoking additional STA tools.
- Our GNN model inspired from path delay calculations reduces the required GNN receptive field and improves model explainability and interpretability.
- We further leverage net and cell delay prediction as auxiliary tasks to facilitate model training, which greatly improves model performance.
- We explore and compare against a deep GNN model and call the attention that suitable deep GNN architectures for EDA tasks are critical yet underexplored.
- Our model evaluated with real-world open-source designs demonstrates the ability to generalize across designs.
- The predicted results correlate strongly with labeled data while being magnitudes faster than routing and STA.
- All data and code are open-sourced for reproducibility[1].

The rest of this paper is organized as follows. Section 2 provides a brief overview of prior work, introduces the vanilla deep GCNII model [8], and generated dataset with open-source designs and tools. Section 3

[†]Equal contribution [*]Corresponding author

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530597

[1]https://github.com/TimingPredict/TimingPredict

details the proposed timing engine inspired GNN model with enhanced training of auxiliary edge delay prediction tasks. Section 4 presents the experimental results. Section 5 concludes the paper.

2 PRELIMINARIES AND RELATED WORK

2.1 Machine Learning for Pre-Routing Timing Prediction

Our problem setup closely follows the prior work of [5], where given the circuit placement result, the objective is to estimate post-routing timing behavior. However, prior work only predicted local delay results such as net delay and net slew using net-based engineered features extracted from placement results. Additional calculations are required through PERT traversals [7] using the machine learning-based net delay models to obtain the circuit global timing metrics. Thus, careful feature engineering and additional invocations of STA are required to obtain accurate arrival time and slack estimations.

In this work, we propose an end-to-end machine learning framework to predict arrival time and slack values at timing endpoints directly. By leveraging GNNs, we remove the need for feature engineering and additional STA analysis.

2.2 Over-smoothing in GCNs and Deep GNNs

Graph neural networks (GNNs) are powerful tools in modeling graph-structured data with increasing popularity and wide adoption in the EDA domain. Since circuit netlist can naturally be represented as graphs, GNNs have been successfully adapted to learning a variety of EDA tasks, including parasitic [18, 23], crosstalk [16], routed wirelength prediction [27], circuit partitioning [20], logic synthesis [28] and placement [15, 21] optimization, and analog sizing [25].

Despite such successful application and adoption in EDA, few works have addressed the severe limitation that conventional shallow GNNs pose. To the best of the authors' knowledge, prior work on EDA graph-based learning seldom stacks beyond four layers of GCN. In a K-layer GNN, each node only has a receptive field of K-hop neighborhood. Thus graph topological information beyond its receptive fields can not be captured with graph convolution. Figure 1 shows an example of the receptive field of 2-layer GNN, where the features of graph nodes beyond 2-hops away can not be aggregated to the node embedding of interest. As a result, shallow GNNs can only encode local graph topology features in node embeddings and cannot express global topological information.

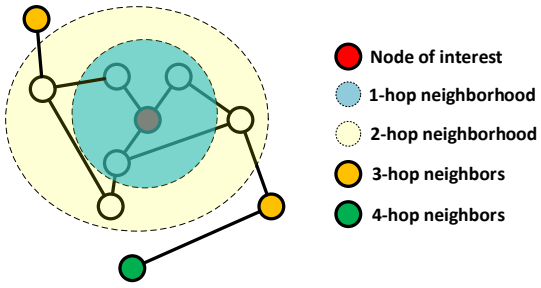

Figure 1: The receptive field of 2-layer GNN. Learned features of only nodes within 2-hops will be aggregated.

However, training deep GNNs is notoriously challenging, and prediction performance is known to decrease with an increasing number of GCN layers. Conventional GNNs suffer from the *over-smoothing* problem, whereby stacking many GCN layers, embeddings of different nodes will share high similarity due to the highly-overlapped receptive fields [14]. Recent works in the machine learning community have developed a series of techniques to relieve the *over-smoothing* issues [9].

In this work, we explore stacking deep GNN layers to learn global timing metrics from timing graphs. Specifically, we establish a baseline deep GNN model from the work of GCNII [8]. Given an undirected graph $G = (V, E)$ with n nodes, let $\boldsymbol{A}$ denote the adjacency matrix and $\boldsymbol{D}$ the diagonal degree matrix. In vanilla GCN [13], the graph convolution layer is defined as follows:

$$\boldsymbol{H}^{(l+1)} = \sigma(\boldsymbol{P}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}), \tag{1}$$

where $\boldsymbol{H}^{(l)} \in \boldsymbol{R}^{n\times d}$ is the node embedding matrix and $\boldsymbol{W}^{(l)} \in \boldsymbol{R}^{d\times d}$ the weight matrix of lth layer, and $\boldsymbol{P}$:

$$\boldsymbol{P} = (\boldsymbol{D} + \boldsymbol{I_n})^{-1/2}(\boldsymbol{A} + \boldsymbol{I_n})(\boldsymbol{D} + \boldsymbol{I_n})^{1/2}. \tag{2}$$

The GCNII model [8] alleviates *over-smoothing* by introducing *residual connections* and *identity mapping*:

$$\boldsymbol{H}^{(l+1)} = \sigma(\underbrace{((1-\alpha_l)\boldsymbol{P}\boldsymbol{H}^{(l)} + \alpha_l\boldsymbol{H}^{(0)})}_{\textit{Residual Connections}}\underbrace{((1-\beta_l)\boldsymbol{I_n} + \beta_l\boldsymbol{W}^{(l)})}_{\textit{Identity Mapping}}), \tag{3}$$

where α_l, β_l are hyperparameters set to 0.1 in our experiments.

Our experimental results in Sec. 4.2 indicate that deep GNN models can have better expressiveness with increased depth but demonstrate poor generalization across different designs, possibly due to over-fitting. We call attention to the EDA community that deep GNN architectures for learning useful and transferable node embeddings with global circuit graph topological information remain challenging and underexplored.

缺点

Table 1: Benchmark statistics. The 21 benchmarks are randomly split into the upper 14 benchmarks for training and the lower 7 for testing.

Benchmark	#Nodes	#Edges		#Endpoints
		Net	Cell	
blabla	55568	39853	35689	1614
usb_cdc_core	7406	5200	4869	630
BM64	38458	27843	25334	1800
salsa20	78486	57737	52895	3710
aes128	211045	148997	138457	5696
wbqspiflash	9672	6798	6454	323
cic_decimator	3131	2232	2102	130
aes256	290955	207414	189262	11200
des	60541	44478	41845	2048
aes_cipher	59777	42671	41411	660
picorv32a	58676	43047	40208	1920
zipdiv	4398	3102	2913	181
genericfir	38827	28845	25013	3811
usb	3361	2406	2189	344
jpeg_encoder	238216	176737	167960	4422
usbf_device	66345	46241	42226	4404
aes192	234211	165350	152910	8096
xtea	10213	7151	6882	423
spm	1121	765	700	129
y_huff	48216	33689	30612	2391
synth_ram	25910	19024	16782	2112
Total Train	920301	660623	608641	34067
Total Test	624232	448957	418072	21977

2.3 Data Generation and Labeling for Open-Source

Open-source has been a huge drive to improve reproducibility and fairness in the field of machine learning research [22], which potentially reduces unintentional errors by obtaining similar results with shared code and data. Fortunately, open-source hardware is gaining momentum in the design ecosystem [1], where hardware designs can be freely used, altered, and distributed. The recent introduction of open-source process design kits (PDKs) [2] and digital flow chains [3], has opened

opportunities for circuit designers and EDA researchers to explore open-source hardware solutions without considering non-disclosure agreement (NDA) restrictions.

In this work, we evaluate our methods using open-source designs, PDK, and EDA tools. Specifically, we generated timing reports on 21 real-world benchmark circuits with OpenROAD on SkyWater 130nm technology. The circuit benchmarks are split into training and testing sets with the benchmark statistics shown in Tab. 1.

3 TIMING ENGINE INSPIRED GRAPH NEURAL NETWORK MODEL

In this paper, we propose an end-to-end GNN model for timing prediction. We predict the pre-routing net delay and pin arrival times without the aid of any external router or timer. We overcome the issue of the limited receptive field by adopting the levelized message-passing flow. Our model has high explainability and interpretability since it closely follows the computation flow of an STA engine.

3.1 Computation Flow of Timing Engines

Although it may be tempting to apply fancy machine learning tricks and brute-force stack deep GCN layers as introduced in Sec. 2.2, the poor generalization results shown in Sec. 4.2 demonstrate the importance of EDA domain knowledge for the successful application of machine learning to our problem. Thus, it is critical to understand the complete delay computation flow in order to design a graph-based model that is both sound and powerful. The pin locations for a net would affect its routing solution, e.g., where Steiner points should be added. For the widely-used Elmore delay model, a net with a complex routing solution would not only have a larger net delay but also introduce a high capacitive load to the driver cell. Such load would be driven by the logic cell from the last level, enlarging the cell delay. An enlarged load would also change the signal transition time (slew). The changes in slew then affect the delay and slew for the next batch of cell arcs. The change and effects are related in a non-linear way, characterized by a non-linear delay model (NLDM) and cell library specific to a design process. A successful model thus has to learn both the routing behavior, net delay model, and delay computation flow in order to accurately predict the slack at timing endpoints.

STA原理

Modern STA engines split the timing analysis process into two steps to break the dependency mentioned above. First, the net delays and net loads are computed based on the net routing. Next, the delay-annotated timing graph is split into topological levels, and the arrival time and slew are computed level by level. This process is called *propagation* which handles the dependency between cell delays and input slew. The number of topological levels is equal to the maximum logic depth, around 300 on large designs with millions of pins. As a result, we need a receptive field of at least 300 hops to precisely model the computation of a timing engine or a conventional GNN of at least 300 layers.

Inspired by the timing engine stated above, we propose re-implement the message-passing model in GNN to reflect the level-by-level propagation. Our specially designed GNN model corresponds to the cell library interpolation and the delay computation process. This combination of timing engine inspirations and graph learning framework yields an explainable, interpretable, and flexible end-to-end timing model.

3.2 Data Representation

Analogous to the two types of timing arcs in timing engines, we represent the circuit as a heterogeneous graph consisting of two types of edges: net edges and cell edges. The nodes in the graph denote pins in the circuit, with features and prediction tasks listed in Table 2. The edge features and tasks are listed in Table 3. The heterogeneous graph structure makes it convenient to develop different message-passing rules for different edge types.

Table 2: Pin features and tasks. EL/RF is short for early/late and rise/fall, i.e., the 4 timing corner combinations in STA.

Type	Name	Size
Features	is primary I/O pin or not	1
	is fanin or fanout	1
	distance to the 4 die area boundaries	4
	pin capacitance	4 (EL/RF)
Tasks	net delay to root pin	4 (EL/RF)
	arrival time	4 (EL/RF)
	slew	4 (EL/RF)
	is timing endpoint or not	1
	required arrival time for endpoints	4 (EL/RF)
Total		27

? model output .sdf ? ✓ ✓ ✓ ✓

Table 3: Edge features and tasks. LUT stands for the look-up tables in the SkyWater130 cell library. For each cell arc, there are 8 LUTs that model the cell delay and slew under 4 timing corner combinations (EL/RF). Early,Late/Rise,Fall

Type	Name	Size
Features	(Net) distances along x/y direction	2
	(Cell) LUT is valid or not	8
	(Cell) LUT indices	$8 \times (7 + 7)$
	(Cell) LUT value matrices	$8 \times (7 \times 7)$
Tasks	(Cell) edge delay	4 (EL/RF)
Total (Net)		2
Total (Cell)		516

different PDK lead to diff LUT?

3.3 Graph Neural Network Model

原理那里有讲这两种哪来的

Similar to the structure of a timing engine, our model consists of two stages: *net embedding* and *delay propagation*.

3.3.1 Net Embedding Model. In this stage, we model the post routing Elmore delay computation from placement inputs. In a router, pins from the same net are connected by wires forming a tree rooted at the net driver pin. The structure of this tree is highly related to all pin locations within the net because Steiner points might be inserted near pin clusters. Thus, we need a model that can extract statistics from all pin locations.

As shown in Figure 2, our net embedding model computes on the bi-direction graph consisting of net edges and reversed net edges. There are three net convolution layers in our model. Each layer transforms the pin features by *graph broadcast* and *graph reduction*. In graph broadcast, information flows from the net driver to the net sinks through the net edges. We concatenate the features of the net driver, the net sinks, and the net edges, and pass through a fully connected neural network layer to obtain the new features of net sinks. In graph reduction, information flows backward from the net sinks to the net driver through the reversed net edges. The model learns statistics from all net sinks through two reduction channels backed by `sum` and `max` operations. We use the node embedding output of the final layer to predict the net delay from inputs to outputs. As a result, our net embedding model can be used standalone to predict net delays similar to the settings of [8]. We also leave free, unsupervised dimensions in the embeddings to represent capacitive load, slew, and other features useful for propagation.

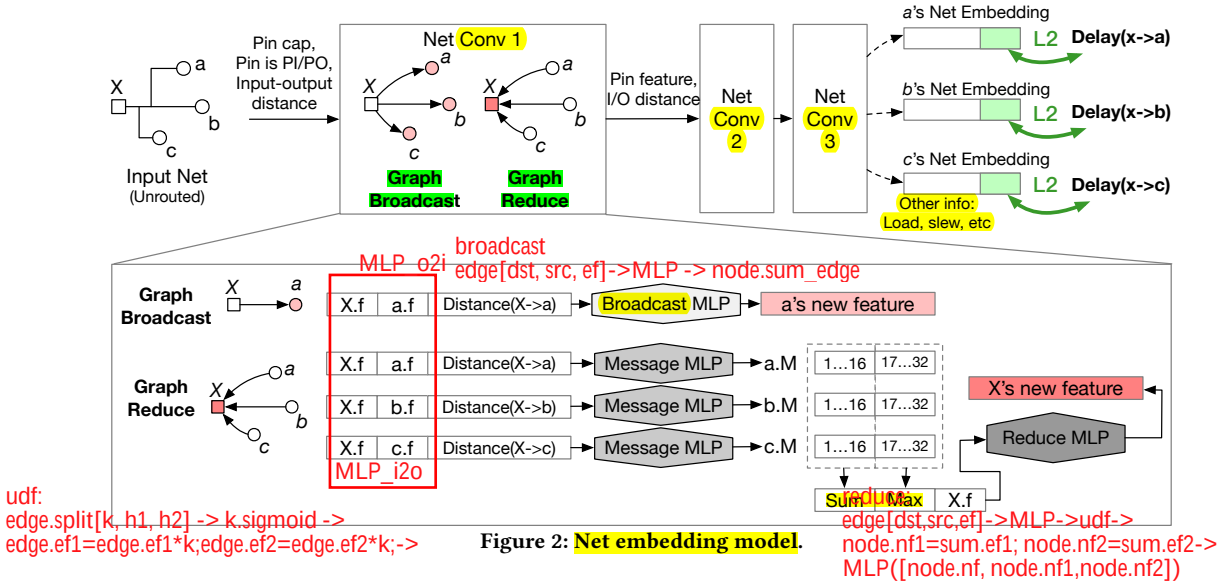

Figure 2: Net embedding model.

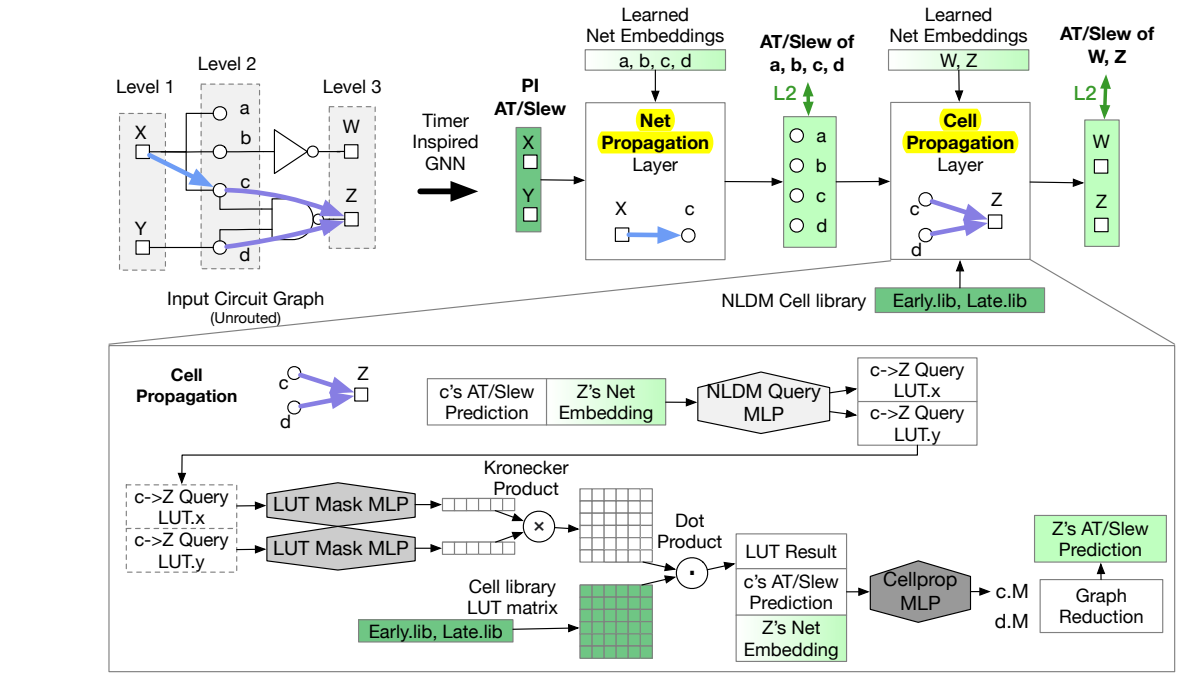

Figure 3: Delay propagation model.

Kronecker https://blog.csdn.net/devil_son1234/article/details/127117818

3.3.2 Delay Propagation Model. This stage models the cell delay interpolation and the level-by-level arrival time and slew computation. Our model design is shown in Figure 3. In this stage, we work on the directed acyclic graph (DAG) formed by the net edges and cell edges. We first compute the topological levels of pins in the DAG. We then propagate the node embedding through each level by alternating between net propagation layers and cell propagation layers.

The net propagation and cell propagation layers are similar to graph broadcast and graph reduction in the net embedding model, respectively. However, cell delay propagation is more complex as it involves cell delay computation with a cell library. The look-up tables (LUTs) in the cell library have high dimensions, up to 512 floating-point numbers for each cell arc. To efficiently learn the cell delays, we develop a LUT interpolation module by first learning the interpolation coefficients separately for two LUT dimensions, and then combining them with a Kronecker product to yield a matrix of coefficients, which is applied to the LUT matrix with the same shape by a dot product. The computed cell arc messages are then fed into two reduction channels with `sum` and `max` operations similar to the net embedding model. This process is analogous to the cell delay lookup process in modern STA engines with the driver cell type and net load statistics as inputs. We use the final node features to predict arrival time and slew for pins.

Compared to the vanilla deep GCNII model of many layers of the net embedding GCN layers in 3.3.1, our model performs only one update for each pin, but the updates are applied asynchronously. The model would learn the computation steps in STA and can be seen as a timing engine learned from data with neural networks as function approximators.

什么意思？

3.4 Enhanced Supervision from Auxiliary Tasks

The objective of our arrival time and slew prediction model can be expressed as minimizing the L2 distance between the predicted arrival time/slew and the ground truth. This loss function is shown in Equation (4), where G is the circuit graph, N is the number of nodes in the graph, AS is the ground truth arrival time/slew of pins, and θ, ϕ are trainable parameters of the net embedding model and the delay propagation model, respectively. This loss function has gradients to both θ and ϕ and can be used to train both models.

$$\mathcal{L}_{atslew}(\theta, \phi | G, AS) = \frac{1}{N} \sum_{\text{node } p \in G} \left\| M^{\phi}_{prop\_atslew}(M^{\theta}_{net}(G)) - AS \right\|_2 \quad (4)$$

In addition to Equation (4), we include edge delay tasks to facilitate model training. Equation (5) shows the cell delay objective to minimize the L2 distance between predicted cell delays and the ground truth CD. This task helps the propagation learning by forcing the model to learn how to compute the cell delay component in arrival times.

$$\mathcal{L}_{celld}(\theta, \phi | G, CD) = \frac{1}{E_{cell}} \sum_{\text{cell arc } e \in G} \left\| M^{\phi}_{prop\_celld}(M^{\theta}_{net}(G)) - CD \right\|_2 \quad (5)$$

Equation (6) shows the net delay objective to minimize the L2 distance between predicted net delays at fan-in nodes and the ground truth ND from routed nets. This guides the net embedding model to learn the routing of nets which is also related to net capacitive loads used in propagation.

$$\mathcal{L}_{netd}(\theta | G, ND) = \frac{1}{N} \sum_{\text{fan-in node } p \in G} \left\| M^{\theta}_{net\_d}(G) - ND \right\|_2 \quad (6)$$

The overall loss function is presented in Equation (7) combining the main task and two auxiliary tasks. The three components are minimized simultaneously.

$$\begin{aligned} \mathcal{L}(\theta, \phi | G, AS, CD, ND) = & \mathcal{L}_{atslew}(\theta, \phi | G, AS) \\ & + \mathcal{L}_{celld}(\theta, \phi | G, CD) + \mathcal{L}_{netd}(\theta | G, ND) \end{aligned} \quad (7)$$

4 EXPERIMENTAL RESULTS

We implement our models using PyTorch and DGL graph learning framework [26]. We train and evaluate our models on a Linux machine with one NVIDIA Titan RTX GPU, two Intel Xeon CPUs at 2.20 GHz, and 256GB RAM. All MLP models used are implemented with 3 hidden layers, each with 64 hidden neurons.

4.1 Net Delay Prediction

We compared our net embedding model for delay prediction in Sec. 3.3.1 with the statistical feature-based model in [5]. We confirm their findings that random forest (RF) performs better than multilayer perceptron (MLP). However, since GNNs can embed local features with larger graph receptive fields than immediate neighbors, our model demonstrates better generalization to test circuits.

4.2 Arrival Time and Slack Prediction

We compare our timer-inspired GNN with the vanilla deep GCNII [8] baseline. We further note that the deep GCNII model runs out of memory after stacking beyond 20 layers, thus we only demonstrate results of stacking 4, 8, 16 layers. Our experimental results and runtime analysis on arrival time and slack prediction performance are shown in Tab 5. To further visualize the performance of our GNN model, Fig. 4 demonstrates the strong correlation in both setup and hold slacks at timing endpoints from our model prediction and ground truth on an example test design `usbf_device`. We summarize our findings below:

Table 4: Comparison on net delay prediction performance (R^2 score) with statistical feature-based ML models [5].

	Benchmark	Statistics-based [5] RF	Statistics-based [5] MLP	Our GNN
Train	`blabla`	**0.9978**	0.9621	0.9918
	`usb_cdc_core`	**0.9905**	0.9339	0.9891
	`BM64`	**0.9968**	0.9604	0.9900
	`salsa20`	**0.9969**	0.9692	0.9863
	`aes128`	**0.9962**	0.9554	0.9770
	`wbqspiflash`	**0.9955**	0.9706	0.9944
	`cic_decimator`	0.9879	0.9331	**0.9946**
	`aes256`	**0.9967**	0.9609	0.9790
	`des`	**0.9966**	0.9678	0.9802
	`aes_cipher`	**0.9907**	0.9223	0.9752
	`picorv32a`	**0.9968**	0.9659	0.9865
	`zipdiv`	0.9926	0.9595	**0.9957**
	`genericfir`	**0.9914**	0.9313	0.9819
	`usb`	0.9958	0.9770	**0.9962**
Test	`jpeg_encoder`	0.9681	0.9624	**0.9739**
	`usbf_device`	0.9388	0.9359	**0.9676**
	`aes192`	0.9630	0.9549	**0.9691**
	`xtea`	0.9235	0.9263	**0.9468**
	`spm`	0.8844	0.8391	**0.9184**
	`y_huff`	0.9390	0.9579	**0.9664**
	`synth_ram`	**0.9761**	0.9735	0.9439
	Avg. Train	**0.9944**	0.9550	0.9870
	Avg. Test	0.9418	0.9357	**0.9552**

- Results demonstrate that the proposed tricks in Sec. 2.2 in GCNII [8] do alleviate *over-smoothing* with 16 layers outperforming shallow networks on training data.
- However, by stacking more GCN layers, the GCNII model does not learn useful graph embeddings and shows poor generalization results for testing circuits.
- Our timer-inspired GNN model can successfully learn both the routing behavior and timing engine computation flow that is transferable to test circuit designs.
- Ablation study on the proposed enhanced supervision of auxiliary tasks in Sec. 3.4 demonstrates that introducing both net and cell delay supervision loss improves the learning results, with net delay loss being more effective.
- Our proposed timing engine inspired GNN model is both accurate and more than three magnitudes faster than running the complete routing and STA in the OpenROAD flow.

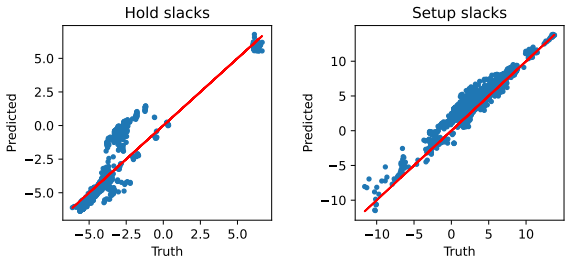

Figure 4: Slack prediction for design `usbf_device`. The result shows a strong correlation between predicted slacks and the ground truth slacks at all timing endpoints.

5 CONCLUSION

In this work, we present a timing engine inspired GNN model to predict the arrival time and slack at timing endpoints. Compared with

Table 5: Comparison on arrival time prediction performance (R^2 score) and runtime (s). We compare with vanilla deep GCNII model [8] using OpenROAD flow as the ground-truth, and conduct ablation study on the edge delay auxiliary tasks.

	Benchmark	Arrival Time / Slack Prediction (R^2 score)							Runtime (s)				
		OpenROAD Flow	Vanilla Deep GCNII [8]			Our Timer-inspired GNN			OpenROAD Flow			Our GNN	
			4 layers	8 layers	16 layers	Full	w/ Cell	w/ Net	Routing	STA	Total	Full	Speed-up
Train	blabla	1.0000	0.5699	0.3878	0.6542	**0.9616**	0.3133	0.8949	836	23.6	859.6	**1.180**	728×
	usb_cdc_core	1.0000	0.3341	-0.1131	0.5273	**0.9751**	0.9529	0.9475	3654	4.7	3658.7	**0.354**	10324×
	BM64	1.0000	0.7404	0.6998	0.7542	**0.9766**	0.9528	0.9146	713	13.6	726.6	**0.496**	1465×
	salsa20	1.0000	0.7624	0.5957	0.7348	**0.9624**	0.1709	0.8943	1738	29.1	1767.1	**0.798**	2213×
	aes128	1.0000	0.7550	0.7039	0.6985	0.8267	0.9081	**0.9487**	1787	51.4	1838.4	**0.739**	2486×
	wbqspiflash	1.0000	0.1052	-0.3237	0.2052	**0.9721**	0.9711	0.9695	181	5.9	186.9	**0.536**	348×
	cic_decimator	1.0000	0.6098	0.2642	0.6704	**0.9840**	0.9692	0.9777	81	3.3	84.3	**0.190**	443×
	aes256	1.0000	0.7939	0.7619	0.7207	0.8488	0.8261	**0.9477**	2886	72.0	2958.0	**0.769**	3845×
	des	1.0000	0.8312	0.8689	0.8101	**0.9922**	0.9879	0.9769	493	16.2	509.2	**0.260**	1957×
	aes_cipher	1.0000	0.6840	0.5846	0.8198	0.9825	0.8825	**0.9828**	1849	18.7	1867.7	**0.359**	5198×
	picorv32a	1.0000	0.6882	0.6403	0.6561	**0.9688**	0.9439	0.9005	987	18.5	1005.5	**0.817**	1230×
	zipdiv	1.0000	-0.3324	-1.0497	0.6560	0.9753	0.9786	**0.9808**	48	3.5	51.5	**0.342**	150×
	genericfir	1.0000	0.8072	0.7132	0.8905	**0.8858**	0.6793	0.8226	409	11.1	420.1	**0.177**	2378×
	usb	1.0000	0.6448	0.2866	0.7364	**0.9784**	0.9645	0.9645	45	3.4	48.4	**0.164**	295×
Test	jpeg_encoder	1.0000	0.7853	0.7259	0.5302	**0.8820**	0.7916	0.8143	4191	70.3	4261.3	**0.773**	5509×
	usbf_device	1.0000	0.7557	0.7412	0.6210	0.9252	0.8838	**0.9330**	1525	21.0	1546.0	**0.557**	2774×
	aes192	1.0000	0.7583	0.7148	0.7462	0.8605	0.7928	**0.9338**	1728	58.6	1786.6	**0.789**	2264×
	xtea	1.0000	-5.1798	-7.0116	-5.4182	**0.9135**	0.8207	0.8772	174	4.9	178.9	**0.544**	329×
	spm	1.0000	0.4056	0.1036	0.4306	0.8975	**0.9157**	0.7224	14	4.2	18.2	**0.087**	210×
	y_huff	1.0000	0.6136	0.5660	0.6017	**0.9256**	0.8568	0.8708	1162	15.4	1177.4	**0.300**	3927×
	synth_ram	1.0000	-4.0511	-1.2762	-8.0822	**0.8656**	0.6437	0.8078	452	9.3	461.3	**0.127**	3635×
	Avg. Train	1.0000	0.5710	0.3586	0.6810	**0.9493**	0.8215	0.9374	1121.9	19.6	1141.6	**0.513**	2361×
	Avg. Test	1.0000	-0.8446	-0.7766	-1.5101	**0.8957**	0.8150	0.8513	1320.9	26.2	1347.1	**0.454**	2664×

prior work, we remove the need to invoke additional STA, demonstrating that our GNN model can learn both routing behavior and timing computation flow in an chitectures and results prone to over-fitting. E designs demonstrate th

[17] DATC-RDF, https://github.com/ieee-ceda-datc/datc-robust-design-flow
[18] DARPA IDEA Program, https://www.darpa.mil/program/intelligent-design-of-electronic-assets
[19] OpenROAD website, https://theopenroadproject.org/
[20] OpenROAD Tool GitHub, https://github.com/The-OpenROAD-Project/OpenROAD
[21] ASAP7 PDK and 7.5-Track Cell Library, https://github.com/The-OpenROAD-Project/asap7
[22] SkyWater 130nm Open Source PDK, https://github.com/google/skywater-pdk
[23] CHIPS (Common Hardware for Interfaces, Processors and Systems) Alliance, https://chipsalliance.org/

REFERENCES

[1] 2021. OpenCores. https://opencores.org/
[2] 2021. SkyWater. https://github.com/google/skywater-pdk
[3] Tutu Ajayi, Vidya A. Chhabria, Mateus Fogaça, et al. 2019. Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project. In *Proc. DAC (DAC '19)*. Association for Computing Machinery, New York, NY, USA, Article 76, 4 pages. https://doi.org/10.1145/3316781.3326334
[4] C.J. Alpert, Jiang Hu, S.S. Sapatnekar, et al. 2004. Accurate estimation of global buffer delay within a floorplan. In *Proc. ICCAD*. 706–711. https://doi.org/10.1109/ICCAD.2004.1382667
[5] Erick Carvajal Barboza, Nishchal Shukla, Yiran Chen, et al. 2019. Machine Learning-Based Pre-Routing Timing Prediction with Reduced Pessimism. In *Proc. DAC*. 1–6.
[6] Wei-Ting J. Chan, Kun Young Chung, Andrew B. Kahng, et al. 2016. Learning-based prediction of embedded memory timing failures during initial floorplan design. In *Proc. ASPDAC*. 178–185. https://doi.org/10.1109/ASPDAC.2016.7428008
[7] Hongliang Chang and S.S. Sapatnekar. 2003. Statistical timing analysis considering spatial correlations using a single PERT-like traversal. In *Proc. ICCAD*. 621–625. https://doi.org/10.1109/ICCAD.2003.159746
[8] Ming Chen, Zhewei Wei, Zengfeng Huang, et al. 2020. Simple and deep graph convolutional networks. In *Proc. ICML*. PMLR, 1725–1735.
[9] Tianlong Chen, Kaixiong Zhou, Keyu Duan, et al. 2021. Bag of Tricks for Training Deeper Graph Neural Networks: A Comprehensive Benchmark Study. *CoRR* (2021). arXiv:2108.10521 https://arxiv.org/abs/2108.10521
[10] Tung-Chieh Chen, Tien-Chang Hsu, Zhe-Wei Jiang, et al. 2005. NTUplace: a ratio partitioning based placement algorithm for large-scale mixed-size designs. In *Proc. ISPD*. 236–238.
[11] Chung-Kuan Cheng, Andrew B Kahng, Ilgweon Kang, et al. 2018. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. *IEEE TCAD* (2018).
[12] Seung-Soo Han, Andrew B. Kahng, Siddhartha Nath, et al. 2014. A deep learning methodology to proliferate golden signoff timing. In *Proc. DATE*. 1–6. https://doi.org/10.7873/DATE.2014.273
[13] Thomas N. Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR* abs/1609.02907 (2016). arXiv:1609.02907 http://arxiv.org/abs/1609.02907
[14] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. 3538–3545.
[15] Yaguang Li, Yishuang Lin, Meghna Madhusudan, et al. 2020. A Customized Graph Neural Network Model for Guiding Analog IC Placement. In *Proc. ICCAD*. 1–9.
[16] Rongjian Liang, Zhiyao Xie, Jinwook Jung, et al. 2020. Routing-Free Crosstalk Prediction. In *Proc. ICCAD*. 1–9.
[17] Yibo Lin, Zixuan Jiang, Jiaqi Gu, et al. 2020. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. *IEEE TCAD* (2020).
[18] Mingjie Liu, Walker J. Turner, George F. Kokai, et al. 2021. Parasitic-Aware Analog Circuit Sizing with Graph Neural Networks and Bayesian Optimization. In *Proc. DATE*. 1372–1377. https://doi.org/10.23919/DATE51398.2021.9474253
[19] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, et al. 2015. ePlace: Electrostatics-based placement using fast fourier transform and Nesterov's method. *ACM TODAES* 20, 2 (2015), 17.
[20] Yi-Chen Lu, Sai Surya Kiran Pentapati, Lingjun Zhu, et al. 2020. TP-GNN: A Graph Neural Network Framework for Tier Partitioning in Monolithic 3D ICs. In *Proc. DAC*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218582
[21] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, et al. 2021. A graph placement methodology for fast chip design. *Nature* 594, 7862 (Jun 2021), 207–212. https://doi.org/10.1038/s41586-021-03544-w
[22] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, et al. 2021. Improving Reproducibility in Machine Learning Research(A Report from the NeurIPS 2019 Reproducibility Program). *Journal of Machine Learning Research* 22, 164 (2021), 1–20. http://jmlr.org/papers/v22/20-303.html
[23] Haoxing Ren, George F. Kokai, Walker J. Turner, et al. 2020. ParaGraph: Layout Parasitics and Device Parameter Prediction using Graph Neural Networks. In *Proc. DAC*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218515
[24] Miodrag Vujkovic, David Wadkins, Bill Swartz, et al. 2004. Efficient Timing Closure without Timing Driven Placement and Routing. In *Proc. DAC*. 268–273. https://doi.org/10.1145/996566.996646
[25] Hanrui Wang, Kuan Wang, Jiacheng Yang, et al. 2020. GCN-RL Circuit Designer: Transferable Transistor Sizing with Graph Neural Networks and Reinforcement Learning. In *Proc. DAC*. Article 201, 6 pages.
[26] Minjie Wang, Da Zheng, Zihao Ye, et al. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315* (2019).
[27] Zhiyao Xie, Rongjian Liang, Xiaoqing Xu, et al. 2021. Net^2: A Graph Attention Network Method Customized for Pre-Placement Net Length Estimation. In *Proc. ASPDAC*. 671–677.
[28] Keren Zhu, Mingjie Liu, Hao Chen, et al. 2020. Exploring Logic Optimizations with Reinforcement Learning and Graph Convolutional Network. In *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*. 145–150. https://doi.org/10.1145/3380446.3430622