Doomed Run Prediction in Physical Design by Exploiting Sequential Flow and Graph Learning

Yi-Chen Lu[1], Siddhartha Nath[2], Vishal Khandelwal[3], and Sung Kyu Lim[1]
[1]School of ECE, Georgia Institute of Technology, Atlanta, GA
[2]Synopsys Inc., Mountain View, CA; [3]Synopsys Inc., Hillsboro, OR
yclu@gatech.edu; siddhartha.nath@synopsys.com; vishal.khandelwal@synopsys.com; limsk@ece.gatech.edu

2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD) | 978-1-6654-4507-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCAD51958.2021.9643435

Abstract*—Modern designs are increasingly reliant on physical design (PD) tools to derive full technology scaling benefits of Moore's Law. Designers often perform power, performance, and area (PPA) exploration through parallel PD runs with different tool configurations. Efficient exploration of PPA is mission-critical for chip designers who are working with stringent time-to-market constraints and finite compute resources. Therefore, a framework that can accurately predict a "doomed run" (i.e., will not meet the PPA targets) at early phases of the PD flow can provide a significant productivity boost by enabling early termination of such runs. Multiple QoR metrics can be leveraged to classify successful or doomed PD runs. In this paper, we specifically focus on the aspect of timing, where our goal is to identify the PD runs that cannot achieve end-of-flow timing results by predicting the post-route total negative slack (TNS) values in early PD phases. To achieve our goal, we develop an end-to-end machine learning (ML) framework that performs TNS prediction by modeling PD implementation as a sequential flow. Particularly, our framework leverages graph neural networks (GNNs) to encode netlist graphs extracted from various PD phases, and utilize long short-term memory (LSTM) networks to perform sequential modeling based on the GNN-encoded features. Experimental results on seven industrial designs with 5:2 train/test split ratio demonstrate that our framework predicts post-route TNS values in high fidelity within 5.2% normalized root mean squared error (NRMSE) in early design stages (e.g., placement, CTS) on the two validation designs that are unseen during training.

I. Introduction

With the burgeoning surge of mobile applications that demand ultra-law latency, building high-performance designs becomes the top priority of most semiconductor companies. To reach the best-achievable signoff timing, designers often perform extensive design space exploration to achieve the best timing closure by running many parallel physical design (PD) implementations, which is highly time-consuming and resource-inefficient because most of the the runs are "doomed to fail" (i.e., not able to meet desired performance). Therefore, to improve the chip design turn-around time (TAT), a method that precisely predicts whether a PD run can successfully achieve the final power, performance, and area (PPA) targets *in early stages of the design flow* is urgently needed.

To solve the above issue, in this paper, we aim to build an end-to-end machine learning (ML) framework that learns to perform doomed run predictions in early PD stages. The term "doomed runs" first originates in [9], which refers to the design implementations that are not able to meet target signoff

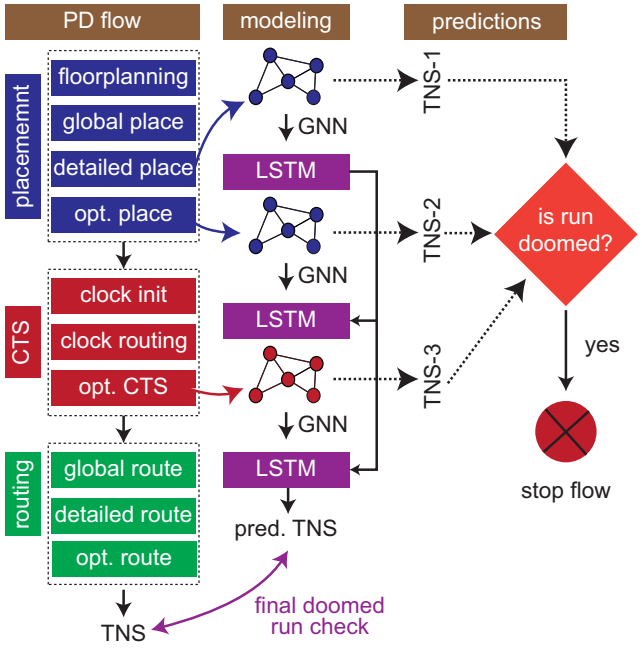

Fig. 1: High-level overview of our sequential modeling approach of PD implementation flow. TNS is selected as the metric for doomed run evaluation. We select three intermediate PD stages from placement and CTS processes to perform sequential modeling. For each intermediate stage, we encode the netlist graph using GNNs and perform per-stage TNS prediction. In addition, we leverage a LSTM network to perform final TNS prediction by taking GNN-encoded features from the three modeling stages as time series inputs.

closures such as timing, power, and design rule violations (DRVs) no matter how much computing resources have been utilized. In this work, to demonstrate the feasibility of the proposed framework, we specifically focus on the aspect of timing, where we take the total negative slack (TNS) value at the post-route stage as the criterion of a successful PD implementation. The goal of this work is to build an accurate post-route TNS predictor using information collected in early stages of the design flow.

Figure 1 demonstrates a high-level overview of the proposed modeling approach using a reference flow of a commercial back-end PD implementation tool. The key idea behind is to model the PD implementation as a sequential process and perform post-route TNS prediction starting from early stages

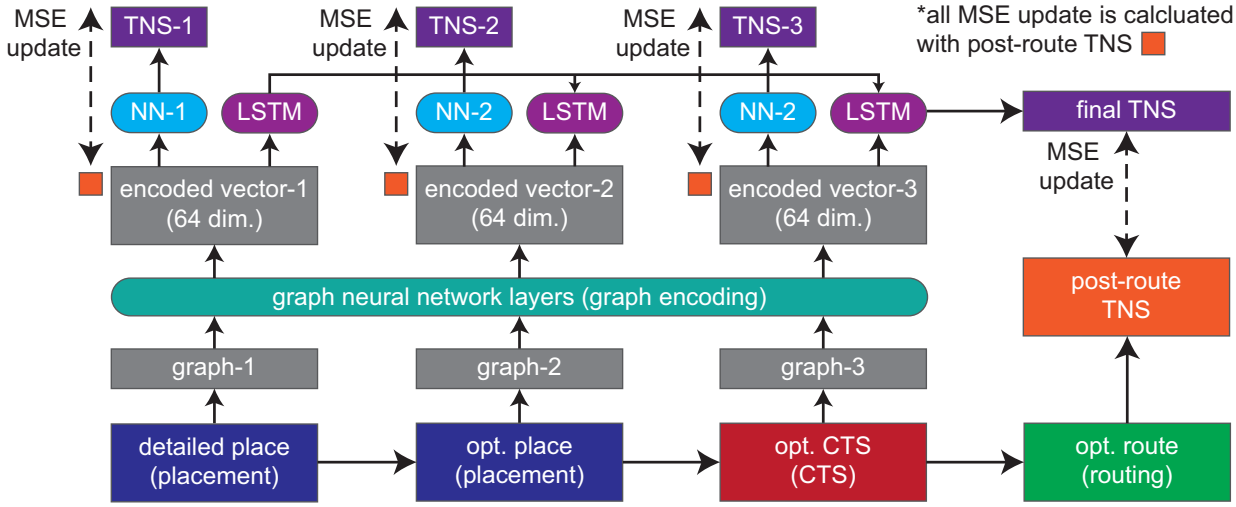

Fig. 2: Overview of our GNN-based LSTM framework. The goal of our framework is to predict the post-route TNS value (colored in orange) in early design stages. by leveraging features extracted from the intermediate stages of the placement and CTS processes. A (global) GNN module is utilized to perform graph encoding across the three stages. The encoded vectors in 64 dimensions are considered as time series data and are taken as the input of the LSTM network. Note that the mean squared error (MSE) loss updates of all predictions (i.e., per-stage prediction and sequential flow prediction) are calculated by taking the post-route TNS values as ground truths.

of the design flow. Therefore, designers can perform early termination of an ongoing PD implementation based on the prediction of our framework. As shown in the figure, our framework is mainly composed of two components: graph neural networks (GNNs) and long short-term memory (LSTM) networks [7], which are responsible for netlist encoding and sequential flow modeling, respectively. The goal of our GNN-based LSTM framework is to predict the post-route TNS value across various PD stages of the design flow that acts as the criterion of a successful PD implementation.

We use a supervised learning framework where we pre-generate a complete dataset with ground truth TNS values of complete PD implementations from seven benchmarks. To ensure the generality of our model (i.e., to apply it successfully on unseen designs), we train a GNN module as a universal graph encoder to embed netlists that come in different sizes and from various stages into meaningful representations in same dimensions. This generalizability of the proposed framework is critical in the realm of EDA, because ML models are practical only if they have the capabilities to perform accurate predictions on unseen benchmarks. These GNN-encoded graph representations are further taken as (1) regular input of the per-stage prediction model and (2) time series inputs of the LSTM model.

The goal of this work is to provide designers a high-fidelity TNS-based doomed run prediction framework for general designs by distilling key design knowledge along the PD flow. Note that our framework does not assume any pre-defined netlist structure, nor the underlying design implementation. Although we use the *Synopsys ICC2* reference design flow, the proposed framework can be extended to other commercial PD tools by using modeling information from their specific flow stages.

II. Related Works of ML in EDA

ML is a promising paradigm that has demonstrated a wild success in the EDA field [8]. ML algorithms powered by deep neural networks (NNs) have shown great promise in PPA prediction in various PD stages such as placement [10, 17], clock tree synthesis [12, 15], routing [14, 22, 24], DRC hotspot prediction [5, 13, 26], IR drop estimation [25], and gate sizing during engineering change order [16] (ECO). Specifically, for post-route timing prediction, the authors of [2] utilize gradient boosting trees to identify the floorplans that are potentially leading to sub-quality timing results, and the authors of [20] also leverage a tree-based method to further perform path-based timing optimization. All of these ML methods harness a rich set of netlist features as inputs that facilitates transfer learning across different designs. Nonetheless, a complete end-to-end ML framework that performs PPA prediction across multiple PD stages is still lacking, which prevents designers from fully exerting the benefits that ML algorithms provide to save design TAT and computational resources. To solve this issue, in this work, we adopt a different modeling approach from previous works by modeling the PD flow as a sequential process. Specifically, we consider features extracted across different PD stages as time series inputs, and leverage a LSTM network to perform cross-stage prediction.

III. Overview of Framework

An overview of our GNN-based LSTM framework is shown in Figure 2. As shown in the figure, in this work, we take three intermediate PD stages from a commercial tool in our modeling. These stages are: “detailed place”, “placement optimization”, and “clock optimization”, where the first two are from the placement process and the last one is from the CTS process. For each targeted modeling stage, we handcraft important

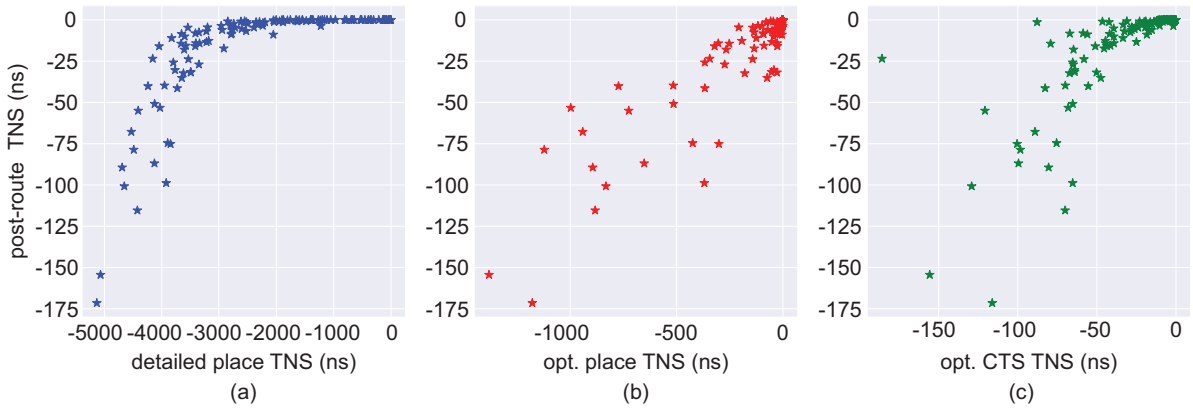

Fig. 3: Correlation analysis on the VGA benchmark. Each dot represents an actual PD implementation. For each targeted modeling stage, we plot the scatter distribution between the estimated TNS values (x-axis) at that stage with the post-route TNS values (y-axis). We observe that as the modeling stage getting closer to the final stage, the fidelity of per-stage TNS estimation (x-axis) becomes higher.

node features that characterize the underlying netlist graph and leverage a GNN module to perform graph encoding. Note that even for a single PD implementation, the netlist graphs across the three modeling stages are dynamically changing because of logic optimization, buffer insertion/deletion, etc. Nonetheless, our GNN encoding is sufficiently expressive to generalize to netlists of various sizes. In the experimental section, we demonstrate that our GNN module has the ability to encode designs with different characteristics into meaningful representations that significantly help the per-stage and sequential flow based TNS predictions.

Since the back-end PD flow is a sequential process, the encoded graph vectors in Figure 2 are highly related to each other. Therefore, we leverage a LSTM architecture to model such dependency in order to accurately predict the final achieved TNS value. Note that the LSTM network use the same parameters to take the encoded features in different time steps as time series inputs, and outputs a single number acting as TNS prediction after three time steps. Finally, with the supervised TNS ground truth obtained after the routing stage, we utilize the mean squared error (MSE) loss to update the parameters in our framework. In Figure 2 we propose an end-to-end framework, which means the parameters from both GNN module and LSTM network can be stored in a single computational graph and be updated jointly by optimizing the loss function (MSE) through gradient descent.

IV. Design of Experiments

Now, we formally define the PD doomed run prediction problem as follows. **Problem: TNS-Based Doomed Run Prediction** Given a RTL with a target synthesis frequency f and a cell density target d, predict whether a PD implementation can successfully achieve post-route TNS value in early stages of the design flow.

A. Database Construction

In this paper, we study supervised learning techniques to solve the doomed run prediction problem. Therefore, pre-generating a representative database is a must in this work. To build the database, we leverage *Synopsys Design Compiler* 2016 to synthesize the netlists from RTL to gate-level designs. Since post-route TNS prediction is the focus of this work, the timing results obtained from the synthesis stage is critical. For each design, we perform experimental sweeps on its synthesis target frequencies to find the maximum frequency that results in a worst negative slack (WNS) greater than zero.

After obtaining the maximum synthesis target frequency of a netlist, we tighten up this frequency target by up to 1GHz as the new frequency target with a step size of 100MHz. For example, assume a design has a synthesis frequency target f, then we will tighten up this frequency target f to get $f + 100$MHz, $f + 200$MHz, ..., and $f + 1000$MHz as the new PD frequency targets of the PD implementations. The rationale behind is that we want to generate a database as diverse as possible in terms of post-route timing results, so that our model would comprehend which netlist features contribute to the success of a PD implementation during the sequential flow modeling process.

Aside from target frequency, routability is also an important factor that affects post-route timing, which is largely determined by the target cell density in early stages. A high cell density target often results in a high congestion during routing, and therefore impacts the timing quality. Following from the frequency setting as aforementioned, for each PD target frequency, we pair it with five different cell density targets: $\{0.7, 0.75, 0.8, 0.85, 0.9\}$. Therefore, for each benchmark, we will generate 50 runs with different pairs of frequency and cell density targets. In total, we have 350 runs across seven designs.

B. Database Analysis

Before diving into the details of our modeling approach, we first perform a detailed analysis of our database, where we take one of our unseen netlists (i.e., not utilized during training), the VGA benchmark, as our case study. Specifically, as the reference flow shown in Figure 1, we take the netlist features in the "detailed place" and "opt. place" stages from

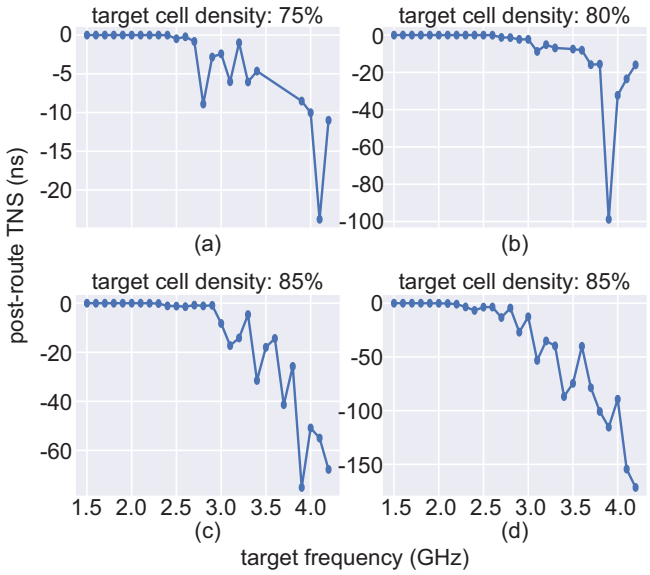

Fig. 4: Frequency sweeping experiments of the VGA benchmark. For each plot, we fix the target cell density whose value is shown on the top of the plot, and sweep around the target synthesis frequency (x-axis). Each dot in the plots represents an actual PD implementation.

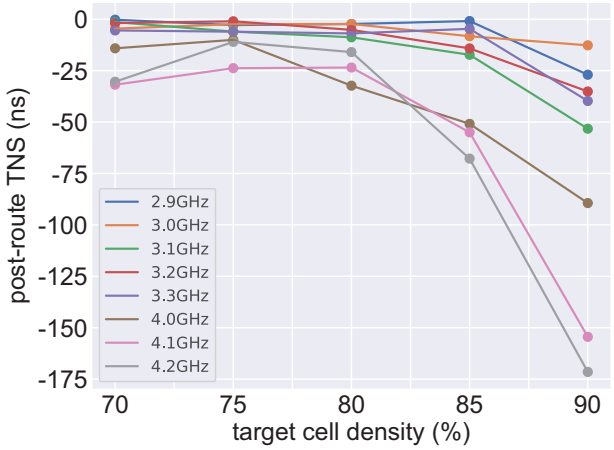

Fig. 5: Target density sweeping experiments on the VGA benchmark. We select eight different target frequencies and sweep around the target cell density (x-axis). We observe that in general, with the same target frequency, the final achieved TNS value degrades as the target cell density increases.

TABLE I: Initial node features for graph representation learning.

name	description
wst slack	worst slack of cell
wst output slew	maximum transition of output pin
wst input slew	maximum transition of input pin
drv net power	switching power of driving net
int power	cell internal power
leakage	cell leakage power

placement, and the features in the "opt. CTS" stage from CTS to perform the modeling. Therefore, in total, our GNN-based LSTM framework comprehends the time series information across three sequential PD stages.

Figure 3 demonstrates a correlation analysis of the three targeted modeling stages in this work to the final post-route TNS values (y-axis). Each dot in the figure represents an actual PD implementation. For each targeted modeling stage, we plot the scatter distribution of the estimated TNS of each stage (x-axis) to the final post-route TNS of the underlying PD implementation. As shown in the figure, we observe that as we move closer to the post-route stage, the commercial tool's pre-route TNS values (x-axis) are more correlated with the post-route TNS values (y-axis). To model the sequential (time-domain dependent) relationship between each PD stage, we use a LSTM network which takes the GNN-encoded features from each stage as its time series input. Detailed algorithms of our framework are discussed in Section VI.

Finally, Figure 4 and Figure 5 demonstrate the target frequency and target cell density sweeping experiments, respectively. In the figures, we observe that although in general the post-route TNS value becomes worse as the target frequency or the target cell density becomes tighter. However, in some situations, the trends become counter-intuitive, which is due to inherent tool noise. For example, in Figure 4(b), the final TNS value becomes better when the frequency is tightened. Therefore, our modeling approach must comprehend such inherent tool noise.

V. Algorithms

Our TNS-based PD doomed run prediction framework is constructed with two main components: the GNN module and the LSTM network, which together form an end-to-end differentiable ML model. The rationale behind the selection of these architectures is two-fold. First, given that netlists are originally represented as hypergraphs where the edge connectivity and cell characteristics contain valuable information, we leverage GNNs to perform meaningful graph encoding with the consideration of such information. Second, based on the fact that PD implementation is actually a stage-by-stage sequential process where the status of the current stage highly depends on the outcomes from the previous stages, we utilize a LSTM network to model such time-series information. In the following sub-sections, we will discuss each component of our framework in detail.

A. Initial Node Features

GNNs are known to perform effective graph representation learning by constructing meaningful node-level or graph-level embeddings that accurately characterize the underlying graphs [23]. In the realm of EDA, previous works [17, 18] have leveraged GNN modules to iteratively transform the feature vector of a node into better representations by considering the cell characteristics and connectivity information of the neighboring nodes. These learned representations are further leveraged to solve the partitioning [18] and the placement optimization [17] tasks.

To successfully apply GNNs on specific EDA tasks, we have to manually define task-related node features that GNN

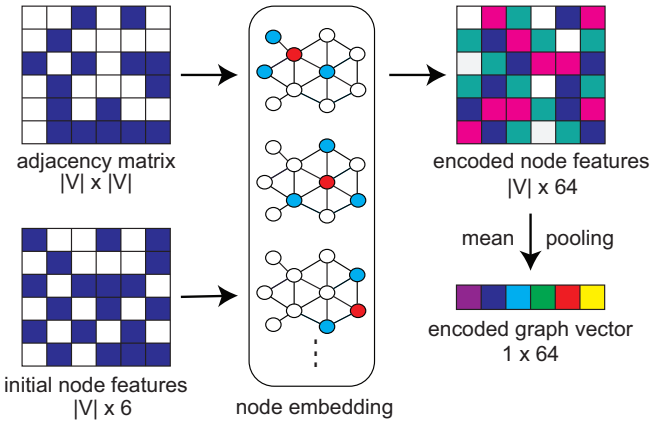

Fig. 6: Illustration of encoding a netlist graph using GNNs. Starting from the initial node features defined manually, a GNN module is used to transform the initial node features into meaning high-dimensional node representations (64 dimensions). Finally, a global graph pooling operation is utilized to transform the node embedding into a single graph vector. This graph vector is further taken as the input of the LSTM network and per-stage NN model as shown in Figure 2.

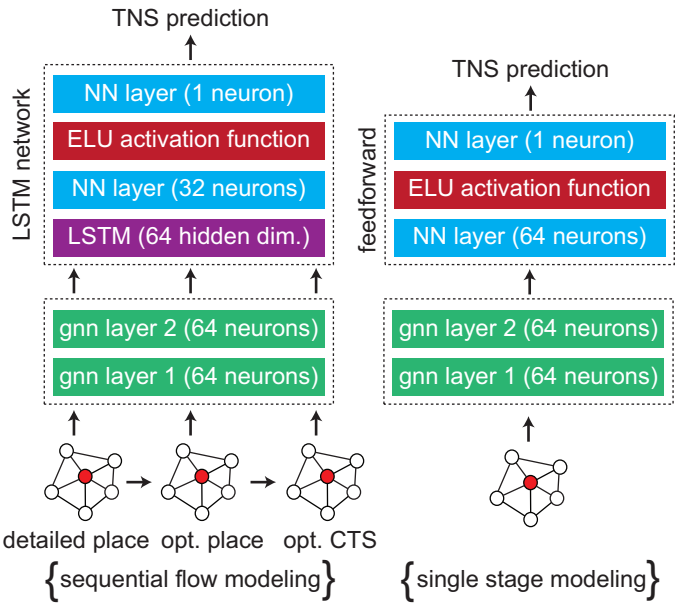

Fig. 7: Detailed architecture of our GNN-based LSTM framework. The proposed framework performs two kinds of modeling approaches with a shared GNN module. The left-hand side is the sequential modeling approach, where the GNN-encoded features across different PD stages are taken as the input of the LSTM network. On the right shows the per-stage modeling approach, where the GNN-encoded features of each stage are taken as the input of a dedicated feed-forward neural network that also performs post-route TNS prediction.

modules can extract insight from. The initial node features we define in this work are shown in Table I. We use domain expertise to extract these features from timing reports, power reports, and technology files. Note that in this work, as shown in Figure 2, our GNN module performs the graph representation learning across various (intermediate) PD stages, which implies we are performing encoding on dynamic graphs. Therefore, the features shown in Table I can be obtained from all the three targeted modeling stages across placement and CTS processes. Based on these initial features, we train our GNN module to obtain meaningful representation in graph-level by performing graph representation learning.

B. GNN as Graph Encoder

Figure 6 shows an illustration of our graph encoding process. As shown in the figure, the goal of graph learning is to transform the initial features defined in Table I into a high-dimensional vector that represents the underlying netlist at a particular intermediate PD stage. Note that for each targeted modeling stage as shown in Figure 2, we perform the netlist graph encoding with the same GNN module (i.e., one GNN module is utilized across all modeling stages). The reason we leverage the same module to encode netlists at different stages rather than developing a per-stage GNN encoder is because we want our framework to be generalizable. Training separate encoding modules for various stages may boost the training accuracy but will lower generalizability and practical adoption of our framework.

As shown in Figure 6, our GNN-based netlist encoding process has two stages. The first stage is termed as node embedding, where for each instance (node) in the design (graph), we transform the manually defined features in Table I into better representations by aggregating local-neighborhood's features. Since in the implementation, our GNN module has 64 neurons at the last layer, each initial feature vector originally in 6 dimensions (Table I) will be transformed into a vector in 64 dimensions. This node representation learning is based on the approach presented in [6]. Given a netlist graph $G = (V, E)$ of an intermediate PD stage, for each node $v \in G$, we first transform the initial node features f_v^0 into embeddings at level $k = K$ as

$$\begin{aligned} f_{N_k(v)}^{k-1} &= reduce\_mean\left(\{\mathbf{W}_k^{agg} f_u^{k-1}, \forall u \in N_k(v)\}\right), \\ f_v^k &= \sigma\left(\mathbf{W}_k^{proj} \cdot \text{concat}\left[f_v^{k-1}, f_{N_k(v)}^{k-1}\right]\right), \end{aligned} \tag{1}$$

where σ denotes the sigmoid function, $N_k(v)$ denotes the neighboring nodes of node v which is limited by the sampling size s_k, W_k^{agg} and W_k^{proj} denote the aggregation and projection matrices at level k respectively, which are learnt by NNs. In the experiments, we set $K = 2$, which means our GNN module has two layers.

Finally, after obtaining the learned node embeddings in 64 dimensions, at the second stage of the GNN encoding process, we perform a global mean pooling over $\{f_v^{k=K}, \forall v \in V\}$ to obtain the final graph vector in 64 dimensions. The encoded graph vectors across various intermediate PD stages are taken as the input to (1) the dedicated model at each stage that performs per-stage TNS prediction and (2) the LSTM network that performs sequential modeling of the PD flow.

C. Sequential Modeling of PD Flow

Given that the PD flow is sequential, we use a LSTM [7] network to perform TNS-based doomed run prediction based

on the features extracted from various stages. The LSTM network is a type of recurrent neural network (RNN) that predicts time-series data using feedback loops by taking the predictions made in previous time steps as the inputs of the current time step. Therefore, it can be considered as a network that unrolls over time based on the length of sequence. In this work, the graph vectors extracted from the three modeling stages as shown in Figure 2 form a sequence of length three. Hence, our LSTM network unrolls three times to make the final TNS prediction.

The LSTM architecture is composed of three gates, which are input gate i, forget gate f, and output gate o. These gate connections are known to facilitate the network to preserve long term "memories" (i.e., information from previous time steps). Given an input sequence x_t at time step t, the gate connections are governed as

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{2}$$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{3}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \tag{4}$$

where $\{W\}$ and $\{b\}$ denote the weights and biases, σ denotes the sigmoid activation function, and h_{t-1} denotes the output from the previous time step, where h is often termed as the hidden state, and is obtained from the cell state c as

$$\tilde{c_t} = tanh(W_c \cdot [h_{t-1}, x_t] + b_c), \tag{5}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c_t}, \tag{6}$$

$$h_t = o_t \odot tanh(c_t), \tag{7}$$

where $\odot$ denotes the element-wise multiplication.

The detailed architecture of our GNN-based LSTM framework is shown in Figure 7. We first leverage GNNs to perform graph encoding across different intermediate PD stages, then we utilize a LSTM network to perform TNS prediction by considering the encoded vectors as time-series inputs. Note that the TNS value prediction is targeted at the post-route stage. The reason we adopt ELU [3] rather than ReLU [1] as the activation function is because ELU provides a smoother non-linearity around zero value and solves the dying ReLU problem. Finally, we want to emphasize that although the proposed framework is composed of two components: the GNN module and the LSTM network, it is end-to-end differentiable. The parameters in both components are stored in the same computational graph and are updated jointly by optimizing the MSE loss of the prediction through gradient descent.

D. Training Methodology

Algorithm 1 presents the training methodology of our GNN-based LSTM framework, where a gradient descent optimizer Adam [11] is utilized to optimize the loss function (MSE) through supervised learning. The procedure of the graph encoding process is shown in Lines 1–9. Note that we utilize the same GNN module to encode netlist graphs taken from different PD stages (Lines 14–16) in order to make the framework generalizable. Finally, after the graph encoding processes, we leverage the LSTM network to perform sequential modeling of the PD flow (Lines 18–20) with an aim to predict the TNS value after the post-route stage.

Algorithm 1 GNN-based LSTM framework training methodology. We use default values of $\alpha = 1e-4, K = 2, \beta_1 = 0.9, \beta_2 = 0.999$.

Input: $\{G_1\}$: netlist graphs at the "detailed place" stage, $\{G_2\}$: netlist graphs at the "opt. place" stage, $\{G_3\}$: netlist graphs at the "opt. CTS" stage, $\{Y\}$: groundtruth TNS values at the post-route stage.

Input: α: learning rate, $\{f^0\}$: initial node features, K: maximum aggregation level, $\{s_k, \forall k \in \{1, ..., K\}\}$: k-hop neighborhood sampling size, $\{W_k, \forall k \in \{1, ..., K\}\}$: parameters of NN at hop (level) k, $\{\theta_L\}$: initial parameters of the LSTM network, $\{\beta_1, \beta_2\}$: Adam parameters.

Output: $\{\hat{y}\}$: post-route TNS predictions.

1: **function** *GNN_Encode*($graph = (V, E)$):
2: $f_v^0 \leftarrow \frac{f_v^0}{\|f_v^0\|_2}, \forall v \in V$
3: **for** $k \leftarrow 1$ to K **do**
4: $N_k(v) \leftarrow$ Sample s_k neighbors at k-hop
5: $f_{N_k(v)}^k =$ reduce_mean $\left(\{\mathbf{W}_k^{agg} f_u^{k-1}, \forall u \in N_k(v)\}\right)$
6: $f_v^k = sigmoid\left(\mathbf{W}_k^{proj} \cdot \text{concat}[f_v^{k-1}, f_{N_v(v)}^k]\right)$
7: $f_v^k \leftarrow \frac{f_v^k}{\|f_v^k\|_2}, \forall v \in V$
8: **return** $v \leftarrow reduce\_mean\{f_v^K \forall v \in V\}$ ▷ encoded graph vector
9:
10: **while** $\{\theta_L, W_k\}$ do not converge **do**
11: $loss \leftarrow 0$
12: **for** (g_1, g_2, g_3, y) in (G_1, G_2, G_3, Y) **do**
13: $g_1' \leftarrow GNN\_Encode(g_1)$ ▷ detail place
14: $g_2' \leftarrow GNN\_Encode(g_2)$ ▷ opt. place
15: $g_3' \leftarrow GNN\_Encode(g_3)$ ▷ opt. CTS
16: $h_0 \leftarrow \mathbf{0}$ ▷ hidden vector initialization
17: $h_1 \leftarrow LSTM(h_0, g_1'; \theta_L)$
18: $h_2 \leftarrow LSTM(h_1, g_2'; \theta_L)$
19: $\hat{y} \leftarrow LSTM(h_2, g_3'; \theta_L)$
20: $loss$ += $(y - \hat{y})^2$
21: $gradient \leftarrow \nabla_{\theta_L, \mathbf{W}}(loss)$
22: $\theta_L, \mathbf{W} \leftarrow Adam(\alpha, gradient, \theta_L, \mathbf{W}, \beta_1, \beta_2)$

TABLE II: Our seven benchmarks and their attributes in *TSMC 28nm*.

Design Name	# Nets	# FFs	# Cells	Usage
JPEG	231,414	37,540	214,666	training
LEON	442,635	108,720	445,381	
ECG	85,058	14,028	84,127	
LDPC	51,534	2,048	48,339	
TATE	206,780	31,416	209,002	
AES	104,704	10,688	114,086	testing
VGA	57,072	17,052	56,897	

VI. Experimental Results and Discussion

In this section, we perform thorough experiments to demonstrate the achievements of our GNN-based LSTM sequential flow modeling framework. We validate our framework on seven industrial designs with a train/test split ratio of 5:2. The characteristics of each benchmark after synthesized under TSMC 28nm technology node using *Synopsys Design Compiler* are shown in Table II. As mentioned in Section IV, to generate the dataset for supervised learning, for each benchmark, we tighten the PD performance target from the maximum frequency obtained from the synthesis stage that achieves

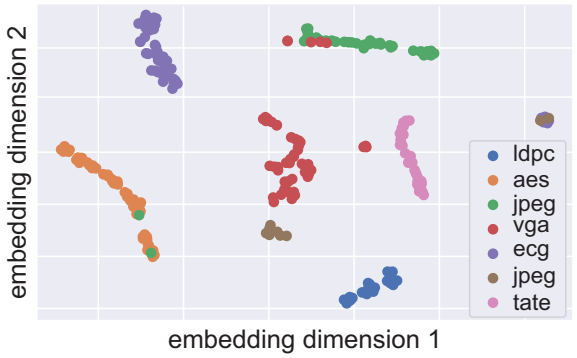

Fig. 8: t-SNE visualizations of the GNN learned node representations. We concatenate the GNN-extracted features from the three targeted intermediate modeling stages, and leverage t-SNE [19], a dimension reduction technique, to plot the GNN-extracted high dimensional features (64*3 dimensions) on a 2D plane. Note that each dot represents a complete PD implementation of the underlying design.

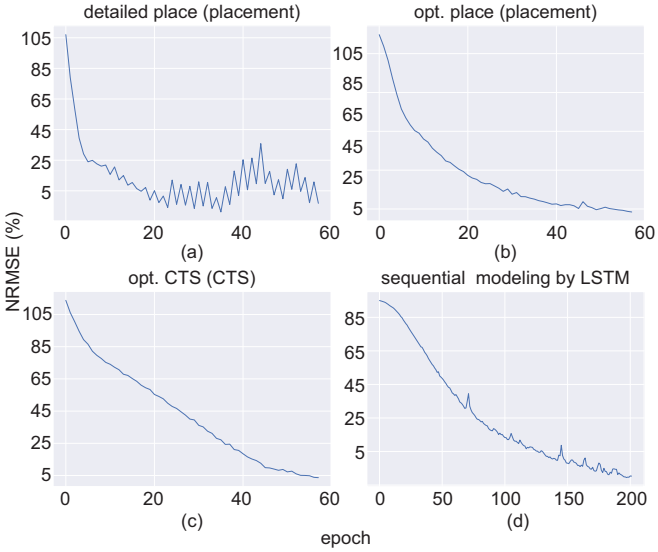

Fig. 9: Training loss iterations of the per-stage TNS prediction and the sequential flow based TNS prediction.

WNS greater than zero. The PD performance tightening is achieved by increasing the obtained synthesis frequency by a maximum value of 1GHz with an interval of 100MHz (i.e., we generate 10 different PD target frequencies by tightening). In addition, we map each target PD frequency with five different target cell densities: {70, 75, 80, 85, 90}%, since the final timing results are greatly affected by routability. Therefore, in total for each benchmark in Table II, we generate 50 different PD runs with various combinations of frequency and cell density targets.

Finally, the proposed GNN-based LSTM framework is implemented in *Python3* with the aid of the *PyTorch* library. Specifically, the GNN implementation is based on the help from the *PyTorch Geometric* [4] library. The seven benchmarks utilized in this work are obtained from *OpenCores.org* and the *ISPD 2012 benchmark suite* [21].

A. GNN Graph Embedding Results

Graph embedding conducted by the GNN module is a highly critical modeling stage in this work, since it enables us to perform transfer learning that facilitates our model to generalize to unseen netlists. To evaluate the effectiveness of graph learning, we leverage the t-distributed stochastic neighboring embedding [19] (t-SNE) dimension reduction technique to visualize the high dimensional encoded graph representations. Specifically, for a PD implementation, we concatenate the graph vector in 64 dimensions of each modeling stage (i.e., forming a 192 dimensions vector), and leverage t-SNE to visualize the encoding in 2D. The result is demonstrated in Figure 8. Note that each dot in the figure denotes a complete PD run. As shown in the figure, we observe that different PD implementations of a same netlist form a self-contained cluster with very few exceptions. This suggests that our GNN module has the ability to differentiate different designs by extracting key netlist features across seen and unseen netlists. Therefore, we have confidence that our framework is generalizable, which is a crucial aspect of the feasibility of ML models in EDA.

B. Per-Stage TNS Prediction

The proposed framework models a PD implementation by extracting netlist features from three intermediate PD stages spanning from placement and CTS. The "per-stage prediction" means that for each modeling stage, we train a neural network to directly predict the post-route TNS value by taking the GNN-encoded features at that stage as inputs. The key reason for conducting the modeling approach is that we expect our framework to have the ability to perform TNS prediction, so that designers can apply it as a doomed run predictor by stopping PD implementations in early stages.

The first three plots in Figure 9 show the training loss iterations in the three modeling stages. "Epoch" on the x-axis represents the number of times that the proposed framework iterates through the whole training dataset. "NRMSE" on the y-axis denotes the normalized root-mean-squared error and is calculated by normalizing the obtained root-mean-squared error (RMSE) that comes with "unit" (e.g., ns) by the difference between the maximum and minimum ground truth values, where the formula is $NRMSE = \frac{RMSE}{y_{max} - y_{min}}$. NRMSE is a popular metric that is utilized to compare prediction results in different scales. In our case, since the post-route TNS distributions of different designs vary greatly, NRMSE is a suitable evaluation metric that helps to evaluate the predictions across various designs. In the figure, we observe that the loss of each modeling stage converges quickly as the training iteration increases. Finally, the validation results are shown in Table III, where we observe that with the modeling stage getting closer to the final stage, the post-route TNS prediction made by per-stage dedicated NN model becomes more accurate.

C. Sequential Flow-Based TNS Prediction

Now, we demonstrate the accuracy of the proposed framework for the sequential modeling approach. Unlike the per-stage prediction approach that directly predicts the post-route

TNS value based on the GNN-encoded vector of that stage using a dedicated NN model, in this experiment, we model the encoded graph vectors across the three targeted modeling stages as time series data, and take them as the inputs of the LSTM network to perform the TNS prediction. The training loss iteration of this experiment is shown in Figure 9(d), where we see that the loss decreases steadily when the training iteration increases. As shown in the figure, the LSTM network is trained with more epochs than the dedicated per-stage NN model, which is because there are more modeling parameters in the LSTM network than in the dedicated models as shown in Figure 7.

The validation results with the sequential modeling approach is shown in Table III, where we observe that the LSTM network predicts the post-route TNS values with high accuracy than the single-stage model. This is largely because the LSTM network leverages a richer set of input features than the dedicated per-stage model by considering the GNN-encoded vectors from all modeling stages as time series inputs. Nonetheless, there exists a trade-off between the modeling accuracy and the runtime of feature collection. Although the predictions in early stages are not as accurate as the predictions that leverages features from later stages of the design flow, our per-stage prediction models still predict the final TNS value with high accuracy because of successfully encoding the netlists by GNN modules from different stages into meaningful representations.

Figure 10 demonstrates scatter and bin-based distribution plots of the predicted and ground truth TNS values on two unseen netlists. Note that each dot in the figure denotes a complete PD implementation. As shown in the figure, although post-route TNS distributions of the two unseen designs vary significantly, our model still has the capability to perform high-fidelity predictions across these two designs. Furthermore, compared with the analysis shown in Figure 3, we conclude that our framework not only makes the predictions in high-correlation but also in high-fidelity. This conclusion is made by observing that the ranges of x-axes in Figure 10 are much closer to the ranges of ground truth TNS values than the ones in Figure 3 which are achieved by tool's estimation. The proposed framework can easily be leveraged by designers to stop ongoing PD runs in early stages of the design flow that are predicted to be doomed (based on the predicted TNS values).

VII. Discussion

Ideally, we want to perform the doomed run prediction as early as possible in the PD flow. However, there is a trade-off between the fidelity of an ML model's prediction and the runtime of its input features collection. With more and more features collected from latter stages of the design flow, ML models are prone to make more accurate predictions. Nonetheless, the runtime of feature collection will increase if more features from late design stages need to be collected. This will make the model not amenable for practical adoption. In this work, we balance this trade-off by performing sequential modeling and confining the framework to collect features up to the routing stage (not included). As shown in Table III, our framework predicts TNS with high accuracy using information from early PD stages. Our framework, therefore, enables designers to terminate a PD run early based on the predictions, which improves productivity and resource usage.

TABLE III: Prediction results on validation designs of our modeling approaches that include three per-stage TNS predictions and one sequential flow based TNS prediction, RMSE denotes root-mean-squared error, NRMSE denotes normalized root-mean-squared error, and CC denotes the Pearson correlation coefficient.

Unseen Designs	RMSE (ns)	NRMSE (%)	CC
per-stage modeling: detailed place (placement)			
AES	0.91	11.2	0.90
VGA	22.2	12.6	0.88
per-stage modeling: place opt. (placement)			
AES	0.88	10.9	0.91
VGA	14.7	8.3	0.91
per-stage modeling: clock opt. (CTS)			
AES	0.54	9.3	0.92
VGA	11.4	6.5	0.94
all-stage sequential modeling using LSTM			
AES	0.47	5.4	0.91
VGA	9.34	5.2	0.95

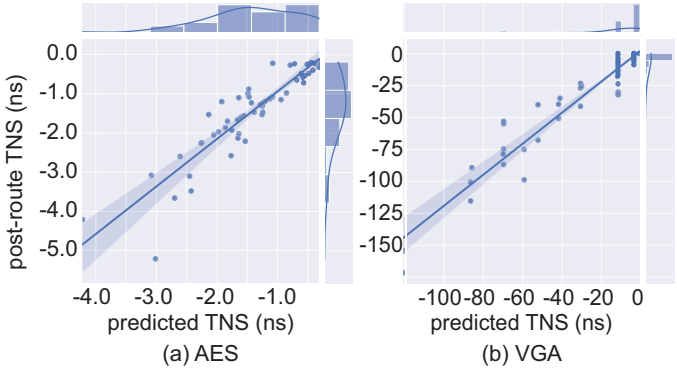

Fig. 10: Scatter and distribution plots of the sequential modeling approach using LSTM on unseen netlists. The regression lines in both plots are generated by fitting least square on the scatter points.

VIII. Conclusion and Future Work

In this paper, we have propose an innovative sequential modeling approach that performs post-route TNS prediction starting from early stages of the design flow. The proposed framework leverages a GNN module to encode netlist features extracted from three intermediate PD stages across placement and CTS. The encoded features are taken as the inputs of per-stage prediction models and a LSTM network that performs sequential modeling of PD implementation. Based on the high-fidelity and high-correlation prediction results achieved, we envision designers to easily leverage the proposed framework to perform PD doomed run prediction and terminate the implementations that are doomed to fail. In the future, we aim to enable the framework to comprehend more doomed run criteria of PD flows and to apply it on more technology nodes.

REFERENCES

[1] A. F. Agarap. Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375, 2018.

[2] W.-T. J. Chan, K. Y. Chung, A. B. Kahng, N. D. MacDonald, and S. Nath. Learning-based prediction of embedded memory timing failures during initial floorplan design. In 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), pages 178–185. IEEE, 2016.

[3] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289, 2015.

[4] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.

[5] H. Geng, H. Yang, L. Zhang, J. Miao, F. Yang, X. Zeng, and B. Yu. Hotspot detection via attention-based deep layout metric learning. In 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pages 1–8. IEEE, 2020.

[6] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In Advances in Neural Information Processing Systems, 2017.

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[8] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong, et al. Machine learning for electronic design automation: A survey. arXiv preprint arXiv:2102.03357, 2021.

[9] A. B. Kahng. New directions for learning-based ic design tools and methodologies. In 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), pages 405–410. IEEE, 2018.

[10] A. B. Kahng. Advancing placement. In Proceedings of the 2021 International Symposium on Physical Design, pages 15–22, 2021.

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[12] S. Koh, Y. Kwon, and Y. Shin. Pre-layout clock tree estimation and optimization using artificial neural network. In Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design, pages 193–198, 2020.

[13] R. Liang, H. Xiang, D. Pandey, L. Reddy, S. Ramji, G.-J. Nam, and J. Hu. Drc hotspot prediction at sub-10nm process nodes using customized convolutional network. In Proceedings of the 2020 International Symposium on Physical Design, pages 135–142, 2020.

[14] R. Liang, Z. Xie, J. Jung, V. Chauha, Y. Chen, J. Hu, H. Xiang, and G.-J. Nam. Routing-free crosstalk prediction. In 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pages 1–9. IEEE, 2020.

[15] Y.-C. Lu, J. Lee, A. Agnesina, K. Samadi, and S. K. Lim. Gan-cts: A generative adversarial framework for clock tree prediction and optimization. In 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 1–8. IEEE, 2019.

[16] Y.-C. Lu, S. Nath, S. S. K. Pentapati, and S. K. Lim. A fast learning-driven signoff power optimization framework. In 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pages 1–9. IEEE, 2020.

[17] Y.-C. Lu, S. Pentapati, and S. K. Lim. The law of attraction: Affinity-aware placement optimization using graph neural networks. In Proceedings of the 2021 International Symposium on Physical Design, pages 7–14, 2021.

[18] Y.-C. Lu, S. S. K. Pentapati, L. Zhu, K. Samadi, and S. K. Lim. Tp-gnn: a graph neural network framework for tier partitioning in monolithic 3d ics. In 2020 57th ACM/IEEE Design Automation Conference (DAC), pages 1–6. IEEE, 2020.

[19] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. Journal of machine learning research, 9(Nov):2579–2605, 2008.

[20] S. Nath and V. Khandelwal. Machine learning-enabled high-frequency low-power digital design implementation at advanced process nodes. In Proceedings of the 2021 International Symposium on Physical Design, pages 83–90, 2021.

[21] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke, and C. Zhuo. The ispd-2012 discrete cell sizing contest and benchmark suite. In Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design, pages 161–164. ACM, 2012.

[22] A. F. Tabrizi, L. Rakai, N. K. Darav, I. Bustany, L. Behjat, S. Xu, and A. Kennings. A machine learning framework to identify detailed routing short violations from a placed netlist. In 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), pages 1–6. IEEE, 2018.

[23] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems, 2020.

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

[25] Z. Xie, H. Li, X. Xu, J. Hu, and Y. Chen. Fast ir drop estimation with machine learning. In Proceedings of the 39th International Conference on Computer-Aided Design, pages 1–8, 2020.

[26] W. Ye, M. B. Alawieh, M. Li, Y. Lin, and D. Z. Pan. Litho-gpa: Gaussian process assurance for lithography hotspot detection. In 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 54–59. IEEE, 2019.