GNN-Based Timing Yield Prediction From Statistical Static Timing Analysis

Chenbo Xi[1], Liang Zhang[1], Biwei Xie[2], Pingqiang Zhou[1]
[1]School of Information Science and Technology, ShanghaiTech University, Shanghai, China
[2]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
{xichb2023, zhangliang2024, zhoupq}@shanghaitech.edu.cn; xiebiwei@ict.ac.cn

2026 31st Asia and South Pacific Design Automation Conference (ASP-DAC) | 979-8-3315-9123-6/26/$31.00 ©2026 IEEE | DOI: 10.1109/ASP-DAC66049.2026.11420486

***Abstract*—As one of the key indicators in digital IC design, timing yield is closely related to the topological correlation among individual timing paths. Timing yield analysis slows down the design cycle, as this time-consuming step needs to be performed repeatedly during post-routing optimization iterations. To improve the design efficiency, in this work, we propose a fast yet accurate GNN-based framework to predict the timing yield for the entire design from SSTA (statistical static timing analysis) results. The experimental results show that our method can achieve a speedup of more than one order of magnitude when compared with the conventional analysis flow.**

***Index Terms*—statistical static timing analysis, timing yield.**

I. Introduction

SSTA (Statistical Static Timing Analysis) has been developed to model the increasing on-chip variations (OCV) under advanced technology nodes [1]–[7]. However, although designers can guarantee that the critical paths satisfy timing constraints with a typical probability of 99.865% by performing SSTA, the timing performance of the entire circuit usually decreases, so designers usually set overly strict constraints on timing paths [8]. To alleviate such overdesign, the *timing yield*, indicating the probability that a design can operate normally under given timing constraints, should be analyzed.

Timing yield analysis is quite different from the SSTA among critical paths—while SSTA provides timing reports for individual paths considering OCV, the timing yield analysis is typically based on SSTA results, and further captures the correlation among multiple timing paths to estimate the yield for the overall design [9]. For example, Fig. 1 illustrates three physical designs, where designers are able to signoff each path at corner states (usually at 3σ condition) using SSTA tools (e.g., PrimeTime [5] and Tempus [6]), indicating that the probability that a path can satisfy timing constraints is no less than 0.99865. However, the timing yield of the entire design could be much worse—for design in Fig. 1a, there is only one path linked to n endpoints, which can also be regarded as n fully correlated paths, and the rate that this design can work under timing constraints is 0.99865; conversely, for design in Fig. 1b with n fully independent paths, the success rate is 0.99865^n, which drops quickly as n grows (for instance, $0.99865^{100} = 0.87$, $0.99865^{1000} = 0.26$). In practical implementations, the paths are usually partially correlated, as demonstrated in Fig. 1c, where the success rate cannot be determined simply.

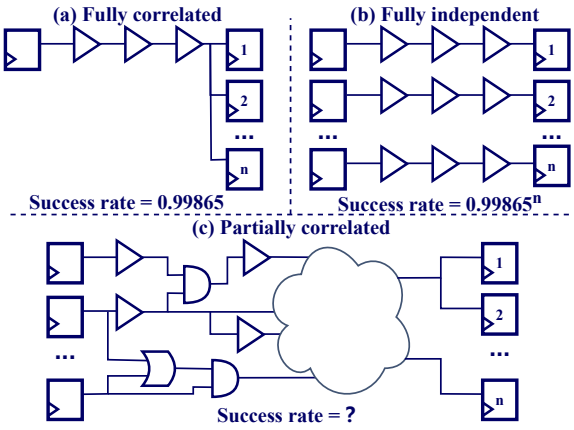

Fig. 1. (a) A design with n fully correlated paths. (b) A design with n independent paths. (c) A design with partially correlated paths.

To perform timing yield analysis for a partially correlated design, a typical process consists of three steps: (1) perform fast GBA (graph-based analysis) SSTA for the entire design; (2) perform accurate PBA (path-based analysis) SSTA for critical and sub-critical paths to get more accurate timing reports for certain paths; (3) analyze the topology correlation among these paths. Advanced commercial EDA tools (such as PrimeShield [10]) perform accurate timing yield analysis (denoted as *design variation analysis* in [10]), by adopting the Monte Carlo analysis to handle the topological correlation [8], [11]. However, as the Monte Carlo method requires a huge number of samples to ensure accuracy, the complexity of design variation analysis could be pretty high; thus, it is usually performed during the post-routing optimization stage. What's worse, multiple optimization iterations are typically required during post-routing, and designers may perform timing yield analysis multiple times for timing-driven optimization, which increases the runtime of each iteration [12].

To improve the design efficiency, in this work, we propose a fast flow to predict the overall timing yield from post-route SSTA results based on GNN (graph neural network). Our work can be essentially helpful in the early iterations of post-routing

optimization during which the efficiency is a major concern. In the literature, several works have explored the application of machine learning for timing analysis [13]–[16]. Specifically, [14], [15] propose learning-based methods to predict PBA results from GBA results, but only for STA (Static Timing Analysis), not SSTA.

Our contributions can be summarized as follows:

- In the literature, we are the first to propose a learning-based framework to predict the timing yield of the entire design from SSTA results. Our work provides a fast timing yield evaluation method during post-routing optimization iterations.
- We first propose a fast GNN-based model to predict the path-based SSTA results from graph-based SSTA results. Then based on the predicted path-based SSTA results, we further develop a GNN-based model to predict the timing yield of the entire design.
- We compare the runtime of our predictive framework with the typical analysis flow (using PrimeTime [5] and PrimeShield [10]), results show that our framework can achieve a speedup of 8.5× on average with high accuracy.

II. Preliminary

A. Timing Analysis: Graph-based vs. Path-based

In physical design, multiple paths may intersect with each other, which complicates accurate timing analysis. The trade-off between accuracy and runtime has given rise to two timing analysis methods: GBA can perform fast analysis, while PBA can reduce the pessimism of timing analysis despite being slower (especially for large circuits). Both methods generate timing reports for individual paths. Under GBA mode, the worst transition time is always propagated. Under PBA mode, the value will be computed based on the real input transition time, but longer runtime is required.

B. SSTA and Variation Modeling

SSTA extends STA by treating delay/transition time as statistical distributions to account for process variations, offering probabilistic timing results [1]–[6].

For advanced technology nodes and very low supply voltages, the moment-based variation modeling method is applied to provide the most accurate timing results [5], [6], which is adopted in POCV/SOCV SSTA through the industry Liberty Variation Format (LVF) [7]. Thus, for delay and transition time, SSTA tools provide mean, standard deviation and corner values, rather than one deterministic value in STA.

C. Timing Yield Analysis and Success Rate

The increasing OCV effect under advanced technology nodes makes SSTA unable to thoroughly gauge the timing performance of the overall design. The key reason is that some cells appear in multiple timing paths simultaneously; thus, analysing paths individually leads to inaccuracy. Typically, timing yield analysis targets on modeling the correlations among paths, which requires the path-based SSTA results as input, since the analysis of correlation is valid and justifiable only based on the accurate results of each individual path.

Commercial EDA tools, such as PrimeShield [10], can perform *design variation analysis* to handle variation correlation among paths, which provides an important reference for the overall timing yield through the $success\ rate$ metric in the report. It should be clarified that the actual yield of a design is usually statistically determined during the binning process after silicon being fabricated, while the $success\ rate$ is a value calculated by PrimeShield, usually in the late stages of the design process, which indicates the probability that a design can operate normally under given design constraints. Since the two metrics are highly correlated, in this work, we regard the $success\ rate$ reported by PrimeShield as the golden result of timing yield analysis and aim to estimate it leveraging GNNs, from post-route SSTA results, which is unexplored in the literature.

D. Graph Neural Networks

GNNs have been shown to perform well in the field of EDA [13], [14], which are effective in solving problems that can be modeled as graphs and the information of nodes is relative to each other. Both of our prediction tasks presented in Section III are suitable for adopting GNNs: for SSTA result prediction, in order to predict timing metrics for a cell in the timing path, the information of its neighbor cells and the connectivity relationship should be considered; for timing yield prediction, in order to model the correlation among paths, their individual timing information and their connectivity relationship should be considered. Furthermore, circuits consist of cells and nets, which can be naturally embedded into graphs.

III. Methodology

Fig. 2 illustrates the complete flow of our timing yield prediction method. Firstly, the netlist files after post-route, design constraint (SDC) files, parasitic (SPEF) files, and libraries are processed by SSTA tools under GBA mode to produce GBA timing results quickly, which are then passed to our proposed PBA predictor to estimate POCV PBA timing results. Finally, taking the predicted PBA results as input, our proposed timing yield predictor generates the estimated design timing yield results.

A. POCV PBA Results Predictor

Our work aims to predict four timing metrics for SSTA: the transition time, the mean value of the delay, the sensit value of the delay, and the corner value of the delay. The input of our POCV PBA predictor is a graph that keeps the connectivity relationship in the original circuit timing path, with features extracted from GBA timing report (generated by commercial EDA tools) and libraries. The output is a graph with the same connectivity and edge features, and four timing metrics for each node (gates in circuits) are predicted.

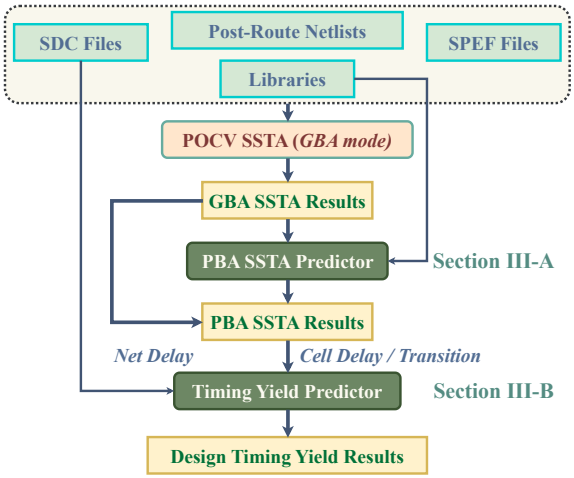

Fig. 2. Timing yield prediction flow.

1) Feature Embedding: GBA timing reports from SSTA tools provide abundant information, but not all of them are helpful to our prediction task. Referring to circuit knowledge and the feature selection in [14], [15], which propose methods to predict metrics for deterministic STA, we carefully select certain features, which are also justified to be important by our parameter-sweeping experiments:

- **Node Features:** *input signal direction, transition ratio, cell input transition, cell output transition, cell mean delay, cell sensit delay, cell corner delay, cell load capacitance, max cell load capacitance, cell functionality.*
- **Edge Features:** *wire signal direction, number of fanouts, wire delay, driver pin transition, receiver pin transition.*

Compared with features adopted in [14], [15], the *mean/sensit/corner delays* are additionally included.

The node feature $transition\ ratio$ is defined as follows: considering a gate i which has multiple inputs, its transition ratio for one specific input j is defined as:

$$transition\ ratio_{ij} = \frac{trans_{ij}}{\max_j(trans_{ij})}, \tag{1}$$

where $trans_{ij}$ denotes the transition time for the timing arc starting from cell j and terminating at cell i.

2) Graph-Learning Model Structure: To predict POCV PBA timing metrics, we present an edge-feature-aware graph attention network, denoted as EGAT. The inputs to our EGAT network are two matrices, node features A and edge features B from a graph, which can be generated by pre-processing the GBA SSTA report (on single path) – extracting the connectivity in the report, and embedding node and edge features mentioned in Section III-A1. Notice that SSTA is performed on individual paths, thus, the generated graph here has a chain structure – each node has at most two neighbors. The output is also a graph, which consists of the same edge features directly from input $timing\ graph\ (GBA)$, and same node features (as they do not change in GBA/PBA reports), but with an update on 4 SSTA timing metrics predicted: *cell output transition, cell mean delay, cell sensit delay, cell corner delay,* which are regarded as the estimation to PBA SSTA results.

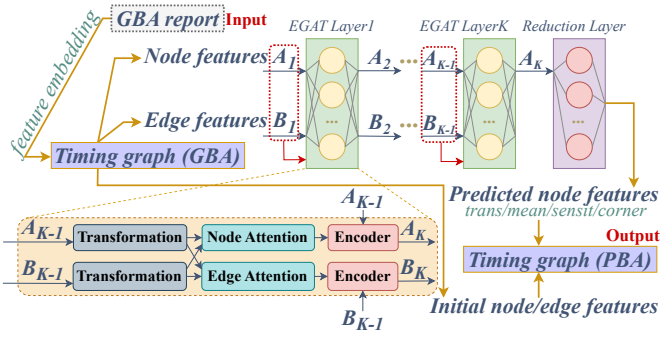

Fig. 3. Architecture of GNN-based PBA predictor. The red blocks indicate the residual connection.

A K–layer network structure is shown in Fig. 3. According to the rules of GNNs, a graph network with K layers enables each node to have a K-hop receptive field. It has been mentioned that too many graph layers may lead to over-smoothing issues [13], and several works in EDA field that use deep graph networks (with ≥ 20 graph layers) try to deal with over-smoothing by leveraging identity matrix or transformer layers [14], [16], which may require extra runtime/memory during training. Luckily, we notice that in the GBA and PBA results, generally, only a portion of the cells have obviously different timing metrics in one path, indicating that we do not need information from faraway nodes, or the entire path, to help predict metrics for a single node. In this work, our model can achieve the highest accuracy when $K = 4$ (see Section IV-B). Compared to the deep graph networks [14], [16], the fewer layers in our model alleviate the over-smoothing problem and reduce time consumption for training.

The GNN layers used in our work are briefly described below (for more details, the readers can refer to [17]). In the transformation blocks, a linear transformation is performed on both matrices of node features and edge features to map them to hidden dimensions. In the attention blocks, the attention mechanism from the transformer [18] is referred, and an attention coefficient for each node/edge is calculated, considering the adjacent nodes and edges simultaneously. Implementations of Encoder and Reduction layer are introduced below:

- **Encoder**: For the K-th EGAT layer, the node features $\mathbf{A}_{K-1}$ and the edge features $\mathbf{B}_{K-1}$ are accepted for residual connection. Denoting the node features from node attention block as G, and edge features from edge attention block as H, we have:

$$\mathbf{A}_K = \mathbf{G} + \mathbf{W}_A^K \mathbf{A}_{K-1}, \mathbf{W}_A^K \in \mathbb{R}^{F_A^K \times F_A^{K-1}}, \tag{2}$$

$$\mathbf{B}_K = \mathbf{H} + \mathbf{W}_B^K \mathbf{B}_{K-1}, \mathbf{W}_B^K \in \mathbb{R}^{F_B^K \times F_B^{K-1}}, \tag{3}$$

  where $\mathbf{W}_A^K$ and $\mathbf{W}_B^K$ are trainable parameter matrices, and F_A, F_B symbolize the dimension of their features.
- **Reduction Layer:** Only the node features $\mathbf{A}_K$ from the K-th graph layer is accepted in the reduction layer.

A multilayer perceptron (MLP) is used to generate the estimated SSTA PBA timing metrics.

Combining the predicted metrics with other initial node/edge features, we can get *timing graph* (PBA), which is the input to our timing yield predictor.

B. Timing Yield Predictor

Timing yield analysis considers the topological correlation among individual timing paths, and depends on accurate timing analysis result of each logical path. Therefore, when estimating the timing yield for the overall design, our yield predictor takes the PBA SSTA results as input.

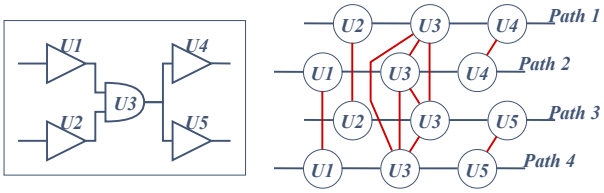

Fig. 4. STA tools provide path-level analysis. In the yield analysis graph, common cells are connected to capture the topology correlation.

1) Pre-processing and Feature Embedding: Fig. 4 illustrates an example of constructing the overall design graph from PBA results. As SSTA tools perform path-level analysis, PBA results contain 4 individual timing paths, all of which are constructed as chained graphs as introduced in Section III-A. $U3$ is marked as a *common cell* for path 2 and 3, as it belongs to both paths. The other cells can be marked respectively. Considering that the key of timing yield analysis is capturing the topological correlation between paths among common cells, we model such correlation by connecting the *common cells* in different paths, as shown in Fig. 4, thus, the individual paths can percept information from the neighbor paths through the connections of common cells during graph learning. The selected node features and edge features for the yield predictor are basically the same as those introduced in Section III-A for the PBA SSTA predictor, while the node feature *transition ratio* is removed.

2) Graph-Learning Model Structure: Taking the design shown in Fig. 4 as an example, the graph-learning structure of its relevant timing yield predictor is shown in Fig. 5. The input *Timing graph (PBA)* consists of multiple chained timing sub-graphs, which can be generated by pre-processing PBA SSTA reports on individual paths, or directly adapting the output from our PBA predictor, as introduced in Section III-A, with additional linkings added on *common cells* (red lines). Similar to the process in Section III-A, the matrices of node features and edge features from *Timing graph (PBA)* are the input to EGAT layers, and the output is another graph G with the same connectivity and processed node features A' and edge features B'. During this process, the features among different paths can propagate through *common cells* linkages so that topology correlation can be captured.

We then predict timing yield using two important designs:

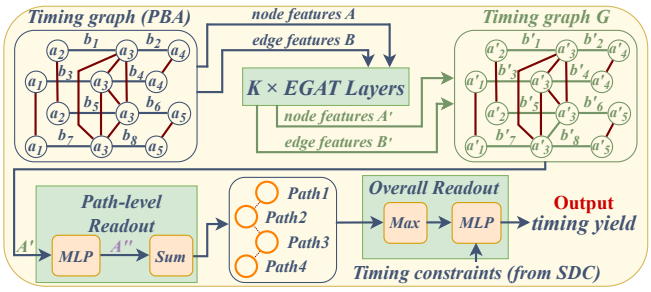

Fig. 5. GNN-based timing yield prediction for the design shown in Fig. 4. a_i/a'_i and b_i/b'_i denote the features of each node and edge in a graph.

- **Path-level Readout:** We first apply an MLP on A' to generate a matrix of node features A''. Then, node features are accumulated along independent timing paths. For example, $U1, U3, U4$ are in timing path 2 in Fig. 4, so we sum up their relevant features in A'' to get only one feature for this path, denoted as the orange circle $Path\ 2$ shown in Fig. 5.
- **Overall Readout:** The path-level readout result will be taken as input, and the SDC file will be referred to, aiming at extracting timing constraints $\vec{l}$, including clock period, clock latency, input delay, output delay, load capacitance, etc. Denoting the path-level readout result as $\mathbf{J} \in \mathbb{R}^{4\times P}$, where P is the dimension of the features, and 4 is for the number of paths, a max operation is performed along each dimension to produce $\vec{r} \in \mathbb{R}^{1\times P}$. Then, denoting the concatenation operation as $||$, the final timing yield metric is estimated by:

$$timing\ yield = MLP([\ \vec{r}\ ||\ \vec{l}\ ]), timing\ yield \in [0, 1]. \quad (4)$$

IV. Results

A. Experimental Setup

We implement our models using PyTorch and DGL framework [22] in Linux environment with Intel Xeon E5-2690 Processor at 2.6GHz. Models are trained on NVIDIA GeForce RTX 2080Ti GPU. Commercial 20+nm technology libraries [7] are used in our work. As the libraries do not include moment-based variation information initially, we sample and simulate to produce LVF libraries using commercial Spice simulator and library characterizer. PrimeTime [5] is used as

TABLE I
STATISTICS OF BENCHMARKS USED FOR TRAINING AND TESTING.

Train	#Cells	#Nets	Test	#Cells	#Nets
AES-Verilog	14,030	14,519	*AES-128*	2,534	2,801
E_MDU	10,328	14,370	*blabla*	9,954	11,901
OpenGFX430	7,346	7,849	*salsaa*	8,977	10,814
A59_decoder	39,962	42,698	*y_huff*	11,172	11,906
TinyMIPS	1,730	1,870	*DataMemory*	16,836	16,891
DataMem	25,757	25,843	*USB_CDC*	1,454	1,622

TABLE II
PERFORMANCE OF PBA SSTA PREDICTORS. MEAN ABSOLUTE ERROR (ps) / MAX ABSOLUTE ERROR (ps) ARE REPORTED.

Benchmarks		AES-128	blabla	salsaa	USB_CDC	y_huff	DataMemory	average
GCNII [19]	*trans*	1.822 / 13.197	1.436 / 10.472	2.035 / 41.984	1.680 / 16.176	1.670 / 23.153	2.256 / 33.804	1.817 / 41.984
	mean	1.620 / 13.380	1.351 / 5.065	1.586 / 320.382	1.831 / 87.209	1.420 / 52.915	1.636 / 97.791	1.574 / 320.382
	sensit	1.594 / 7.429	1.484 / 5.824	1.447 / 37.404	1.560 / 8.403	1.429 / 25.267	1.617 / 19.620	1.522 / 37.404
	corner	1.604 / 5.663	1.149 / 3.862	1.147 / 206.669	1.604 / 40.717	1.333 / 62.059	1.451 / 42.602	1.381 / 206.669
GraphSage [20]	*trans*	1.858 / 6.415	1.320 / 5.950	1.723 / 25.953	1.587 / 7.636	1.676 / 9.447	2.050 / 38.405	1.702 / 38.405
	mean	1.438 / 3.640	1.177 / 3.019	1.326 / 10.951	1.323 / 4.932	1.324 / 6.046	1.535 / 15.641	1.354 / 15.641
	sensit	1.519 / 3.996	1.242 / 3.567	1.406 / 15.017	1.372 / 4.471	1.347 / 8.812	1.609 / 20.447	1.416 / 20.447
	corner	1.436 / 3.671	1.230 / 2.859	1.343 / 9.436	1.313 / 3.994	1.312 / **5.923**	1.529 / 25.005	1.645 / 25.005
GAT [21]	*trans*	1.907 / 9.534	1.238 / 11.464	1.555 / 34.865	1.472 / 34.648	1.589 / 33.997	1.569 / 50.382	1.555 / 50.382
	mean	1.491 / 5.685	1.352 / 3.523	1.314 / 11.884	1.295 / 10.408	1.316 / 12.590	1.411 / 24.552	1.363 / 24.552
	sensit	1.452 / 7.781	1.169 / 3.502	1.281 / 31.711	1.299 / 14.321	1.283 / 17.539	1.337 / 47.347	1.303 / 47.347
	corner	1.486 / 7.255	1.219 / 3.280	1.337 / 15.176	1.282 / 11.443	1.335 / 14.452	1.413 / 31.503	1.345 / 31.503
Ye's [14]	*trans*	1.521 / 6.141	1.221 / 4.512	1.398 / 12.919	1.444 / 9.062	1.456 / 10.061	1.785 / 16.926	1.471 / 16.926
	mean	1.265 / 3.505	1.106 / 2.792	1.230 / 13.542	1.255 / 10.281	1.248 / 6.172	1.445 / 6.835	1.258 / 13.542
	sensit	1.322 / 3.765	1.299 / 3.276	1.264 / **10.713**	1.247 / 8.141	1.232 / **4.809**	1.487 / 10.047	1.309 / **10.713**
	corner	1.273 / 3.702	1.189 / 2.459	1.275 / 12.570	1.288 / 6.774	1.230 / 6.453	1.456 / 9.915	1.285 / 12.570
Ours ($K = 2$)	*trans*	1.399 / 5.230	1.159 /3.439	1.301 / 19.376	**1.331** / 9.701	1.303 / 10.756	1.371 / 18.136	1.311 / 19.376
	mean	1.261 / 3.056	1.090 /1.778	1.182 /8.842	1.233 / 6.404	1.173 / **5.976**	1.241 / 13.244	1.197 / 13.244
	sensit	1.229 / 3.078	1.076 / **1.824**	1.192 / 24.985	1.267 / 4.738	1.194 / 13. 051	1.248 / 12.593	1.201 / 24.985
	corner	1.217 / 2.920	1.121 / 2.046	1.202 / 15.997	1.215 / 4.882	1.188 / 7.222	1.282 / 7.099	1.204 / 15.997
Ours ($K = 4$)	*trans*	**1.362 / 3.618**	**1.144 / 2.967**	**1.244 / 11.083**	1.335 / **8.907**	**1.248 / 7.119**	**1.308 / 11.121**	**1.273 / 11.121**
	mean	**1.179 / 2.408**	**1.057 / 1.595**	**1.133 / 4.847**	**1.153 / 3.055**	**1.139** / 7.727	**1.136 / 4.603**	**1.132 / 7.727**
	sensit	**1.189 / 2.577**	**1.065** / 1.973	**1.142** / 13.357	**1.177 / 3.772**	**1.160** / 5.212	**1.170 / 4.429**	**1.150** / 13.357
	corner	**1.185 / 2.777**	**1.049 / 1.675**	**1.133 / 8.251**	**1.153 / 3.701**	**1.106** / 6.757	**1.134 / 4.060**	**1.127 / 8.251**
Ours ($K = 6$)	*trans*	1.733 / 7.040	1.253 / 3.764	1.490 / 30.725	1.381 / 9.674	1.476 / 9.383	1.642 / 13.692	1.496 / 30.725
	mean	1.429 / 5.771	1.152 / 3.090	1.292 / 15.766	1.312 / 8.521	1.304 / 7.569	1.508 / 27.156	1.333 / 27.156
	sensit	1.542 / 6.521	1.279 / 3.560	1.407 / 29.895	1.393 / 5.295	1.380 / 17.545	1.585 / 59.906	1.431 / 59.906
	corner	1.335 / 47.542	1.179 / 2.880	1.320 / 34.649	1.311 / 6.752	1.359 / 7.819	1.375 / 11.568	1.313 / 34.649

STA tool to generate timing analysis results. PrimeShield [10] is used to generate the design $success\ rate \in [0, 1]$.

B. Predict PBA Results from GBA Results

12 open-source designs are collected from GitHub [23] and OpenCores [24], and are processed by commercial EDA tools to generate PBA/GBA SSTA reports. For each design, 20,000 timing paths are collected. We train the model on *AES-Verilog, E_MDU, OpenGFX430, A59_decoder, TinyMIPS, DataMem*, and validate it on the other 6 benchmarks.

Table I shows the information of the benchmarks. We utilize Mean Squared Error (MSE) as the loss function. Our model is trained for 20,000 epochs before evaluation, while in each epoch, 16 cases are sampled. Four metrics in PBA results, $Trans$, $Mean$, $Sensit$, and $Corner$ are predicted as individual tasks. We compare the performance of our model with GCNII [19], GraphSage [20] and GAT [21]. [14], [15] predict PBA results from GBA results without considering the OCV. Since [14] (referred to Ye's method) has shown to be more accurate over [15], we also compare the performance of our predictor with Ye's method (with SSTA-related features provided), which contains 100 GNN layers. Compared with GCNII, GraphSage and GAT, our EGAT considers edge features. Compared with Ye's method [14], the key differences in our EGAT method are: 1) *transition ratio* is additionally included in node features; 2) less GNN layers ($K = 4$) are used. We use mean absolute error and max absolute error as evaluation metrics [14], [15], compared with results obtained from PrimeTime [5].

Table II shows the prediction results on 6 test benchmarks. Experimental results indicate that our model achieves the highest performance with exactly 4 EGAT layers. Our model also improves runtime efficiency, as fewer layers reduce training and inference time (compared to the deep graph network implementation). Using a single GPU, it takes more than 24 hours for Ye's model to converge, while our model takes about 4 hours. Generally, our method is accurate with less runtime cost, indicating that our model and feature selection are ideal for the prediction of PBA SSTA results.

C. Predict Timing Yield Metrics from PBA Results

We collect 4,729 designs from two open-source datasets [25], [26], with the maximum number of leaf cells reaching 1,789,191. For [26], we only collect designs that have $advanced$ or $expert$ complexity and $rank$ no less than 18, and similar operations are performed on [25] to guarantee that the collected designs are relatively complex. Same designs in both datasets will be collected only once.

Typically, in timing yield analysis, only critical and sub-critical paths are analyzed to accelerate the process [10]. In our experiment, for each design, we considered the 10 worst paths and set the target success rate at 99.865%. Designs are adjusted so that there will be a violation for the critical path under 3σ state, while the $mean$ delay can meet the target timing constraints, such that the success rate of each design is more evenly distributed in $[0, 1]$. Under such experimental settings, for most of the designs, the success rate of the critical path is greater than 0.9, while the success rate of the overall

TABLE III
PERFORMANCE OF TIMING YIELD PREDICTORS. MEAN ABSOLUTE ERROR (MAE) (PS) / R^2 SCORES ARE REPORTED.

Metrics	GCNII [19]	GraphSage [20]	GAT [21]	Ours(from GBA)	Ours($K = 2$)	Ours($K = 4$)	Ours($K = 6$)	w/o linkage	w/o readouts
MAE / R^2	0.054 / 0.887	0.050 / 0.910	0.064 / 0.847	0.053 / 0.893	0.047 / 0.921	**0.043 / 0.943**	0.055 / 0.882	0.862 / 0.727	0.100 / 0.638

TABLE IV
THE RUNTIME OF THE TRADITIONAL FLOW (PRIMETIME + PRIMESHIELD) AND OUR PREDICTION FLOW.

Flow	Runtime Breakdown (s)				Speedup
PrimeTime + PrimeShield	GBA *30.377*	PBA 114.392	DVA 177.812	/	1×
Ours	GBA 30.377	PBA_{pred} 1.592	DVA_{pred} 2.056	$Process$ 4.084	**8.465**×

design can be lower due to topology correlation among paths. All designs are randomly divided into a training set and a test set in a 7:3 ratio. Our model takes about 2 hours to converge.

In this experiment, the input PBA SSTA results are directly collected from PrimeTime [5], not predicted by our PBA predictor. Several comparisons are included in Table IV-C:

- Results indicate that our model achieves the best performance when $K = 4$.
- Denoting our model ($K = 4$) without $common\ cells$ linkages (red lines in Fig. 4) as *w/o linkage*, and that replacing path-level and overall readout (introduced in Section III-B) by a max operation as *w/o readouts*. Results indicate that these designs are essential to our model architecture.
- GCNII [19], GraphSage [20], and GAT [21] are compared, all of which also apply the $common\ cells$ linkage, path-level readout, and overall readout blocks; otherwise, the performance will deteriorate drastically. Results show that our model can achieve higher accuracy.
- Denoting the prediction directly from GBA results as $Ours(from\ GBA)$. Results indicate the necessity to predict yield based on PBA results, because the accuracy decreases when predicting timing yield from GBA results.

D. Predict Timing Yield Metrics from GBA Results

As experimental results in Section IV-B and IV-C have demonstrated that our PBA predictor and timing yield predictor exhibit the best performance, here we predict timing yield directly from POCV GBA results (generated by PrimeTime), leveraging two predictors. The same dataset setting as that used in Section IV-C was employed. The pre-trained PBA predictor model from Section IV-B is further trained for 5,000 epochs on our training set (introduced in Section IV-C) by transfer learning (which takes less than an hour), and the pre-trained timing yield predictor model from Section IV-C is directly applied. We first predict PBA results from the input GBA reports, then estimate $success\ rate$ (generated by PrimeShield) based on predicted PBA results.

Compared with the golden results from PrimeShield, our method, which consists of a PBA predictor and a yield predictor, achieves an accuracy with a mean absolute error of 0.050, and an R^2 score of 0.914. Compared with the results of $Ours(K = 4)$ in Table III, the mean error increases slightly, while the R^2 score decreases slightly, possibly due to the accumulation of errors among the two predictors. Compared with the results of $Ours(from\ GBA)$ in Table III, the accuracy of timing yield prediction based on estimated PBA results is still higher than the prediction directly from PrimeTime GBA reports – the mean error of $success\ rate$ can be reduced by 5.7% on average, while only requiring an additional runtime that is typically less than one second.

The runtime comparison results are shown in Table IV, where DVA denotes design variation analysis in PrimeShield, PBA_{pred} and DVA_{pred} denote our PBA predictor and timing yield predictor, respectively. $Process$ denotes the runtime for data pre-processing before using the PBA predictor (generate graph from GBA report) and the timing yield predictor (add $common\ cells$ linkages, combine sub-graphs of individual timing paths). The runtime of 15 designs with the largest scale is summed up for evaluation. The runtime of the traditional flow, which is GBA→PBA→DVA, is compared with our proposed prediction flow, which is GBA→Data process→PBA prediction→Data process→timing yield prediction. In our framework, the runtime bottleneck primarily stems from the GBA SSTA phase. Overall, our proposed framework can achieve $8.5\times$ speedup on average, compared with the traditional process.

Although our method can obtain good prediction results at a lower runtime cost, there is still a certain gap from the golden *design variation analysis [10]* in terms of accuracy. We would like to emphasize that the motivation of our work is to provide a method that helps designers perform fast timing yield evaluation, which could be beneficial for improving the overall design efficiency and accelerating the design optimization iteration.

V. CONCLUSIONS

In this work, we explore the accuracy-runtime tradeoff in timing yield analysis and present a fast timing yield prediction framework, which estimates the timing yield metric from SSTA results. First, we propose a GNN-based PBA SSTA predictor. Then we propose another GNN-based predictor, estimating the timing yield metric that cannot be directly formulated by timing analysis. In our future work, we will explore the transferability to other technology nodes.

ACKNOWLEDGMENTS

This work was supported by the Science and Technology Commission of Shanghai Municipality (STCSM) under Grant 24JD1402500. We would like to thank Cheng Li and UniVista Industrial Software Group Co., Ltd. for their support.

REFERENCES

[1] H. Chang and S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single pert-like traversal," in *International Conference on Computer Aided Design (ICCAD)*, 2003, pp. 621–625.

[2] A. Devgan and C. Kashyap, "Block-based static timing analysis with uncertainty," in *International Conference on Computer Aided Design (ICCAD)*, 2003, pp. 607–614.

[3] Y. Zhan, A. Strojwas, X. Li, L. Pileggi, D. Newmark, and M. Sharma, "Correlation-aware statistical timing analysis with non-gaussian delay distributions," in *Proceedings. 42nd Design Automation Conference (DAC)*, 2005, pp. 77–82.

[4] S. Takahashi, Y. Yoshida, and S. Tsukiyama, "A gaussian mixture model for statistical timing analysis," in *2009 46th ACM/IEEE Design Automation Conference (DAC)*, 2009.

[5] Synopsys. Primetime. [Online]. Available: http://www.synopsys.com/implementation-and-signoff/signoff/primetime.html

[6] Cadence. Tempus. [Online]. Available: http://www.cadence.com/content/cadencewww/global/enUS/home/tools/digital-design-and-signoff/siliconsignoff/tempustiming-signoff-solution

[7] J. Zhou, L. Huang, H. Xia, Y. Cai, L. Jin, X. Shi, W. Xing, T.-J. Lin, and L. He, "Lvf2: A statistical timing model based on gaussian mixture for yield estimation and speed binning," in *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*. New York, NY, USA: Association for Computing Machinery, 2024.

[8] Synopsys, "Revolutionizing pre-silicon design robustness analysis with li ding," http://www.synopsys.com/blogs/chip-design/pre-silicon-design-robustness-analysis-discussion-li-ding.html.

[9] M. Pan, C. Chu, and H. Zhou, "Timing yield estimation using statistical static timing analysis," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2005, pp. 2461–2464 Vol. 3.

[10] Synopsys. Primeshield. [Online]. Available: http://www.synopsys.com/implementation-and-signoff/signoff/primeshield.html

[11] Synopsys. Improving design robustness with primeshield: A discussion with li ding. [Online]. Available: http://www.synopsys.com/blogs/chip-design/improving-design-robustness-primeshield-discussion-li-ding.html

[12] N.-Z. Lee, V. N. Kravets, and J.-H. R. Jiang, "Sequential engineering change order under retiming and resynthesis," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 109–116.

[13] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin, "A timing engine inspired graph neural network model for pre-routing slack prediction," in *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, New York, NY, USA, 2022, p. 1207–1212.

[14] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi, "Graph-learning-driven path-based timing analysis results predictor from graph-based timing analysis," in *2023 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2023, pp. 547–552.

[15] A. B. Kahng, U. Mallappa, and L. Saul, "Using machine learning to predict path-based slack from graph-based timing analysis," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018, pp. 603–612.

[16] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi, "Fast and accurate wire timing estimation based on graph learning," in *2023 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2023, pp. 1–6.

[17] H. C. Jun Chen, "Edge-featured graph attention network," *arXiv:2101.07671*, 2021.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[19] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

[20] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 1025–1035.

[21] A. C. A. R. P. L. Y. B. Petar Veličković, Guillem Cucurull, "Graph attention networks," *arXiv:1710.10903*, 2017.

[22] Z. Y. e. a. Minjie Wang, Da Zheng, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv:1909.01315*, 2019.

[23] Github. [Online]. Available: http://github.com

[24] Opencores. [Online]. Available: http://opencores.org

[25] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, "Benchmarking large language models for automated verilog rtl code generation," 2022. [Online]. Available: https://arxiv.org/abs/2212.11140

[26] B. Nadimi, G. O. Boutaib, and H. Zheng, "Pyranet: A large scale hierarchical verilog dataset," 2024. [Online]. Available: https://arxiv.org/abs/2412.06947