NUA-Timer: Pre-Synthesis Timing Prediction Under Non-Uniform Input Arrival Times

Ziyi Wang[1], Fangzhou Liu[1], Tsung-Yi Ho[1], David Z. Pan[2], Bei Yu[1]
[1]The Chinese University of Hong Kong [2]The University of Texas at Austin

***Abstract*—Accurate and swift pre-synthesis timing estimation is crucial for early-stage timing optimization and design space exploration. Recent advances in machine learning have shown significant promise in improving pre-synthesis timing prediction accuracy. However, existing learning-driven methods have overlooked the complexities introduced by the trending hierarchical design paradigm, specifically non-uniform input arrival times (NUIAT). In this paper, we present NUA-Timer, a novel pre-synthesis timing prediction framework designed to address the unique challenges posed by NUIAT in hierarchical timing prediction. To capture the complex long-range timing dependencies under varying NUIAT, NUA-Timer employs a novel bidirectional propagation neural network (BPN), which enables the quantification of timing dependencies using a correlation matrix. Furthermore, we introduce a tailored loss function that leverages post-synthesis critical path labels, thereby aligning the correlation matrix with actual post-synthesis timing dependencies. Comprehensive experiments on both synthetic and open-source designs demonstrate the superiority of our method compared to the state-of-the-art (SOTA) pre-synthesis timing evaluators.**

I. Introduction

Logic synthesis is a crucial phase in the electronic design automation (EDA) flow, where high-level descriptions are transformed into netlists that can be implemented on integrated circuits. One of the primary challenges in this phase is optimizing the timing performance of the circuits. Traditional approaches rely on post-synthesis analysis to refine timing, which is costly as it requires repetitive efforts. To mitigate this issue, there is a growing emphasis on integrating pre-synthesis timing evaluation techniques, which estimate post-synthesis timing metrics before the synthesis process. This enables rapid feedback and facilitates early timing optimization within the design cycle [1].

As integrated circuits continue to grow in size and complexity, the EDA paradigm is shifting from flat to hierarchical design [2]. In this hierarchical approach, a circuit is segmented into multiple blocks at various levels, as illustrated in Fig. 1. Different blocks undergo parallel optimization before integration, enabling more manageable design complexity and facilitating reuse and maintainability. However, this shift introduces new challenges for pre-synthesis timing prediction. When evaluating the timing performance of a specific block, such as an intellectual property (IP) block, the unknown application context makes fixed input arrival time assumptions unreliable [3]. In fact, arrival times often differ across input ports, a condition termed non-uniform input arrival times (NUIAT). As illustrated in Fig. 1, this arises because input signals propagate through different upstream paths, experiencing varying delays.

Recent advances in machine learning have shown significant potential in improving pre-synthesis timing prediction [4]–[7]. These works initiate the process by transforming the register transfer level (RTL) circuit into an intermediate graph format. Following this transformation, they conduct either path-based prediction with feature engineering [4], [5] or graph-based analysis using graph neural networks (GNN) [6], [7]. Despite these advancements, a common limitation among existing learning-driven methods is their foundational assumption that all input signals have zero initial arrival time. This assumption is problematic in hierarchical designs, where upstream delays and interconnect effects cause significant variations in input arrival times.

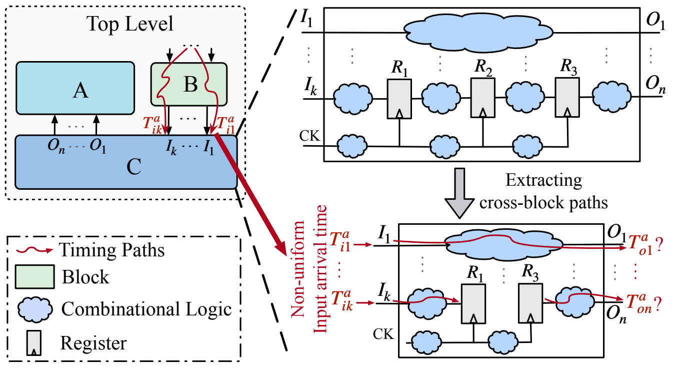


Fig. 1 Illustration of hierarchical timing analysis, which focuses on timing paths across blocks and considers non-uniform input arrival times.

Variations in input arrival times can cause fluctuations in the distribution of pre-synthesis critical paths, directly impacting the synthesis process and the post-synthesis timing characteristics. As shown in Fig. 2, post-synthesis critical paths may vary with different input arrival time conditions. Accurately capturing the post-synthesis critical path distribution is crucial to effectively evaluating post-synthesis timing. However, the dynamic nature of input arrival times and complex circuit dependencies make this challenging. We summarize two unique challenges below.

Firstly, accurately annotating post-synthesis critical paths on the pre-synthesis circuit is challenging due to the lack of a one-to-one mapping between pre- and post-synthesis circuit components. Existing methods either overlook critical path information [6], [7] or depend on pre-synthesis critical paths [4], [5], which often diverge substantially from post-synthesis critical paths. For instance, the startpoint of a pre-synthesis critical path for an endpoint may be a don't-care term to the endpoint, as shown in Fig. 2. Consequently, predictions based on such pre-synthesis critical paths may result in considerable errors.

Secondly, it requires the ability to effectively capture the long-range dependencies between circuit inputs and outputs, which remains a hurdle for existing graph learning methods. Traditional message passing neural networks (MPNNs) [8], [9] are constrained by their localized aggregation scheme, which inherently limits their capability to learn long-range dependencies. While Graph Transformers [10]–[12] are adept at capturing global information within graphs, they lack explicit mechanisms to identify and prioritize critical paths, which is crucial for timing prediction. The inability to accurately model long-range timing dependencies limits the effectiveness of current graph-based timing prediction methods.

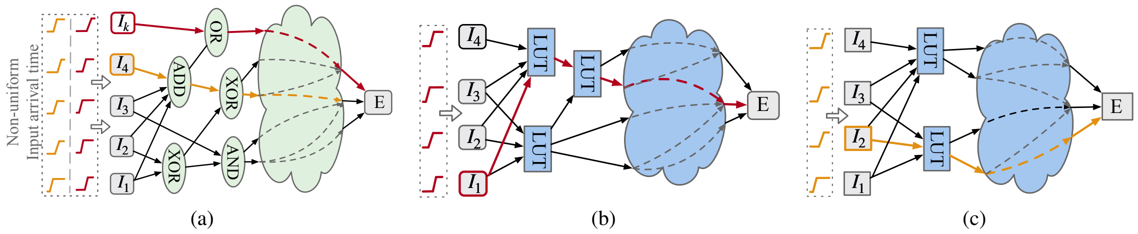


Fig. 2 An example to show how post-synthesis critical paths can differ from pre-synthesis critical paths and may vary with different non-uniform input arrival times (NUIATs). Here, the input I_k is a don't care term of endpoint E. (a) highlights pre-synthesis critical paths, each color per a specific NUIAT scenario. (b) and (c) display the resulting post-synthesis netlists and critical paths under two distinct NUIAT conditions.

To address these challenges, we introduce NUA-Timer, a pre-synthesis timing prediction framework specifically designed for non-uniform input arrival times. We propose a novel bidirectional propagation neural network (BPN) to effectively capture the long-range timing dependencies for each endpoint. This capability is crucial for understanding and predicting the intricate dynamics of circuit behavior under varying input conditions. To enhance the accuracy of our framework, we have also designed a unique loss function that directly leverages post-synthesis critical path information, thereby enabling the BPN model to adaptively capture the distribution of critical paths as input arrival times change.

The major contributions of this paper are listed as follows:

- We introduce a novel Bidirectional Propagation Neural Network that enables the quantification of the long-range timing dependencies associated with each endpoint.
- We develop a global path embedding method that captures post-synthesis critical path information, enabling more accurate pre-synthesis timing prediction.
- We design a customized loss function, using pairwise annotation of post-synthesis critical paths, that guides the BPN to accurately model the relationship between input arrival times and post-synthesis timing behavior.
- We conduct comprehensive experiments on both synthetic and open-source designs to demonstrate the superiority of our proposed method over previous state-of-the-art (SOTA) pre-synthesis timing prediction methods.

The rest of the paper is organized as follows: Section II introduces the problem definition and the background of hierarchical timing analysis, Graph Neural Networks, and pre-synthesis timing prediction. Section III overviews our proposed framework. Section IV details our methods, including aspects such as the preprocessing process, the Bidirectional Propagation Neural Network, and the customized loss function. Section V presents experimental results, followed by the conclusion in Section VI.

II. Preliminaries

A. Hierarchical Timing Analysis

As design evolution continues, there is a notable escalation in the size and complexity of designs. To effectively manage this growth and bridge the productivity gap, key strategies such as IP reuse and hierarchical design have become essential [13]. Hierarchical partitioning allows for the decomposition of large-scale integration designs into smaller, manageable blocks. These blocks, often consisting of duplicated IPs, can be developed in parallel, significantly enhancing design efficiency.

In the realm of chip design, static timing analysis (STA) is crucial as it has a direct influence on the design cycle time. As chip designs become increasingly complex and expansive, conducting a full-chip flat timing analysis has become a burdensome task, both in terms of time and resources consumed. In response to this challenge, the EDA industry is shifting towards hierarchical timing analysis in alignment with the trend of hierarchical design [2], [13]–[16].

As shown in Fig. 1, hierarchical timing analysis primarily concentrates on timing paths that cross hierarchical boundaries while paying less attention to those confined within individual blocks. It usually begins by extracting the cross-block timing paths, forming a reduced circuit design. The arrival time at an output of a block is determined by the arrival times at all the correlated inputs of the block and the maximum delays from all the correlated inputs to the output. When characterizing the timing model of a block, especially an IP block, the application context is unknown [3]. Consequently, it is prudent to avoid making assumptions about the arrival times at the inputs. This highlights the critical need for accurate timing estimation under non-uniform input arrival times.

B. Pre-Synthesis Timing Prediction

In recent years, machine learning methods have been introduced to enable early timing predictions throughout the design flow, from logic synthesis [4]–[7] to physical design [17]–[26] stages. In particular, pre-synthesis timing prediction aims to estimate post-synthesis timing metrics before the logic synthesis stage, allowing for early optimization and reducing the need for iterative design refinements. Existing methodologies can be divided into two distinct categories: path-based [4], [5] and graph-based approaches [6], [7]. Initially, both approaches convert the RTL design into a detailed bit-level graph representation, such as an and-inverter-graph (AIG). This transformation is crucial as it facilitates an intermediate transition from the RTL design to the post-synthesis netlist, thereby bridging the gap between these two stages. Subsequently, different methodologies are employed to extract features for timing endpoints.

In path-based methods [4], [5], the process begins with the extraction of pre-synthesis paths for each timing endpoint from the built graph. For instance, MasterRTL [4] focuses on identifying a single critical pre-synthesis path for each endpoint, whereas RTL-Timer [5] incorporates additional random paths to enhance the robustness of the analysis. Following path extraction, these methods employ extensive feature engineering to meticulously select key features from these paths, which are instrumental in making precise timing predictions. However, the path-based methods may suffer from performance degradation when the pre-synthesis critical path distribution does not align with the post-synthesis one.

Conversely, graph-based methods [6], [7] concentrate on capturing the structural information embedded within the fan-in cone of each endpoint. Specifically, LSTP [7] utilizes a specialized asynchronous GNN that processes the graph in a topological order. This approach

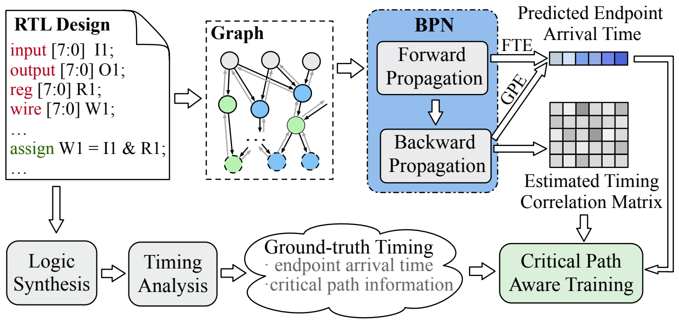


Fig. 3 Overview of NUA-Timer, where FTE and GPE represent forward timing embedding and global path embedding. The critical path aware training aligns the estimated timing correlation matrix with post-synthesis timing-critical dependencies.

allows for the automatic generation of endpoint embeddings through the graph encoder (e.g., GNN), which are subsequently utilized to predict timing performance. Nevertheless, existing graph-based methods follow a localized message-passing scheme, which limits their ability to capture long-range timing dependencies across the entire circuit.

C. Graph Neural Network

Graph neural networks (GNNs) have emerged as a robust approach for analyzing graph-structured data and extracting valuable insights from interconnected information [27], [28]. Among the prevailing GNN architectures are the message passing neural network (MPNN) and the graph Transformer.

MPNNs employ an iterative message passing and aggregation process to capture the structural relationships within node neighborhoods [8], [9]. However, MPNNs are limited in learning long-range dependencies due to their localized message passing approach, restricting information propagation to immediate neighbors and gradual spread across layers. In response to the challenge of capturing long-range dependencies, Graph Transformers have been introduced [10]–[12], extending the Transformer [29] architecture to graph data. By incorporating a self-attention [29] mechanism, Graph Transformers can assign varying importance to nodes relative to each other, enhancing the model's ability to capture distant relationships. Additionally, Graph Transformers can leverage positional encodings [30] to provide the model with knowledge of node positions within the graph topology, further enhancing its capacity to capture long-range dependencies. Despite these strengths, Graph Transformers lack the explicit mechanisms to identify, prioritize, and analyze critical paths through the graph. The implicit nature of path information makes it difficult for Graph Transformers to effectively learn the path-dependent delays that are essential for accurate timing analysis.

D. Problem Definition

Problem 1 (Pre-synthesis Timing Prediction under Non-Uniform Input Arrival Time)**.** *Given a pre-synthesis RTL block, we conduct hierarchical timing evaluation by focusing on the timing paths within the block. Specifically, we consider the timing paths from a primary input (PI) of the block to a register input port or from a PI to a primary output (PO). Our goal is to accurately predict the post-synthesis arrival time at each endpoint (register input port or PO) under varying input arrival time patterns.*

III. Overview

Pre-synthesis timing prediction with non-uniform input arrival times presents significant challenges, primarily in capturing long-range timing dependencies and reflecting the impact of varying input arrival patterns on critical path distribution. To address these challenges, we introduce NUA-Timer, a novel pre-synthesis timing prediction framework.

Fig. 3 shows the overview of NUA-Timer. Our approach begins by transforming the RTL design into a bit-level heterogeneous graph representation, as detailed in Section IV-A. At the core of NUA-Timer lies the Bidirectional Propagation Neural Network (BPN), which performs both forward and backward message propagation on the constructed graph. This bidirectional scheme allows the model to capture intricate long-range timing dependencies between inputs and endpoints, overcoming the limitations of localized message passing in MPNN.

The forward phase (detailed in Section IV-B) allows arrival time to propagate from the PIs to the timing endpoints, generating a forward timing embedding (FTE) for each endpoint. An attention-based aggregation scheme drives this process, capturing local timing dependencies between nodes and their immediate predecessors. Upon completion of the forward phase, the process transitions to the backward phase, where the information flows from endpoints back to inputs. During this stage, localized timing dependencies are propagated backward to capture long-range timing dependencies throughout the entire circuit. These dependencies are encapsulated within a timing correlation matrix, which is then utilized to generate a global path embedding (GPE) for each endpoint. The GPE effectively aggregates timing-critical information across the entire circuit, ensuring a comprehensive representation of the circuit's timing dynamics under non-uniform input arrival times.

By combining FTE with GPE, our framework generates a comprehensive final embedding for each timing endpoint. This combined embedding is then processed through a Multi-Layer Perceptron (MLP)-based regressor, which transforms these embeddings into predicted post-synthesis endpoint arrival times.

During the model's training phase, a customized loss function (Section IV-D) is employed to ensure the alignment of the predicted timing correlation matrix with actual post-synthesis critical path labels. This supervision guides the BPN to adaptively model the dynamic critical path distribution influenced by non-uniform input arrival times.

The detailed descriptions of the algorithms can be found in the following sections.

IV. Methodology

A. Preprocessing

Our framework begins by transforming the input RTL block design into a bit-level graph representation. The input design first undergoes preprocessing to extract the timing paths (combinational logic) from PIs of the block to register input ports or POs. Register input ports in the preprocessed design are treated as POs of the block. The goal of NUA-Timer is to predict the arrival time at each timing endpoint (PO). Following the timing path extraction, a directed heterogeneous graph is constructed from the preprocessed design. This graph considers two types of logic operators: single-bit-output gates such as AND gates, and multi-bit-output arithmetic modules like adders. In this graph, each output bit of these operators is represented as a node, with directed edges connecting input bits (output bits of predecessor operators) to output bits.

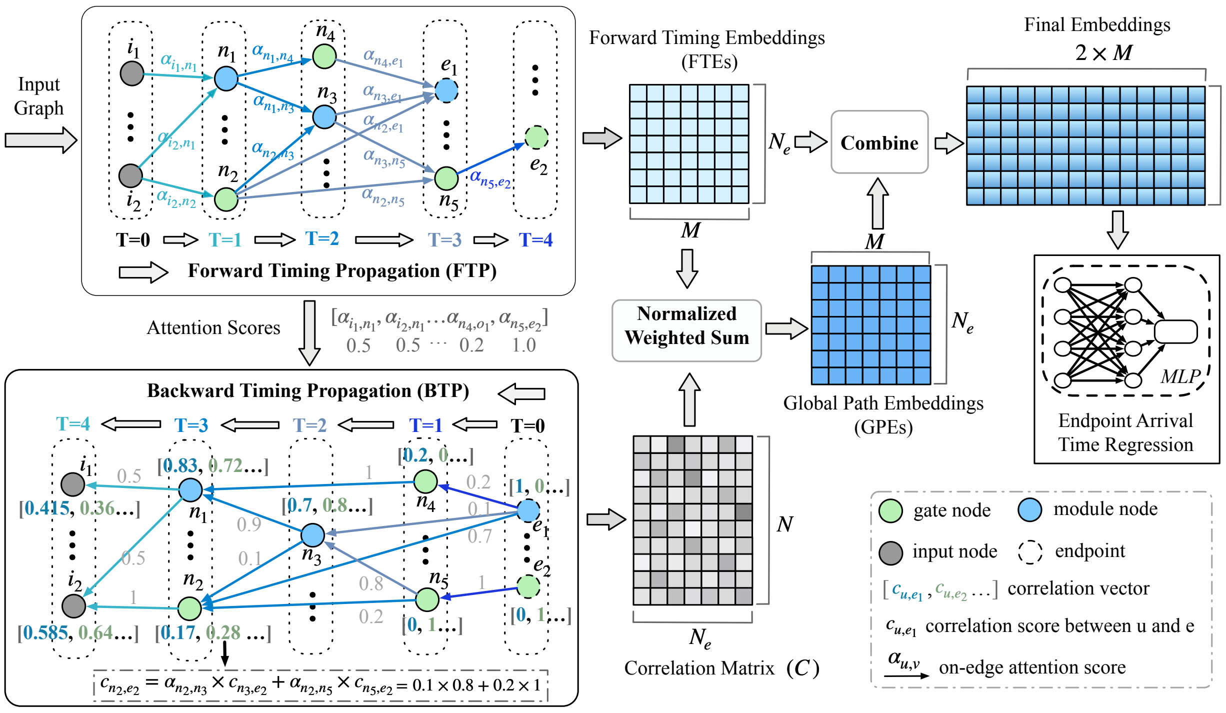

Input Graph
Forward Timing Propagation (FTP)
T=0 T=1 T=2 T=3 T=4
Forward Timing Embeddings (FTEs)
Combine
Final Embeddings
$2 \times M$
N_e
M
Normalized Weighted Sum
Global Path Embeddings (GPEs)
Endpoint Arrival Time Regression
MLP
Attention Scores
$[\alpha_{i_1,n_1}, \alpha_{i_2,n_1} \ldots \alpha_{n_4,o_1}, \alpha_{n_5,e_2}]$
0.5 0.5 ⋯ 0.2 1.0
Backward Timing Propagation (BTP)
$c_{n_2,e_2} = \alpha_{n_2,n_3} \times c_{n_3,e_2} + \alpha_{n_2,n_5} \times c_{n_5,e_2} = 0.1 \times 0.8 + 0.2 \times 1$
Correlation Matrix (C)
N
gate node
module node
input node
endpoint
$[c_{u,e_1}, c_{u,e_2} \ldots]$ correlation vector
c_{u,e_1} correlation score between u and e
$\alpha_{u,v}$ on-edge attention score

Fig. 4 Workflow of Bidirectional Propagation Neural Network (BPN). The forward stage propagates the input arrival times from PIs to endpoints, and captures local timing correlations between nodes and their immediate predecessor. Then, in the backward stage, information flows from endpoints back to PI to capture long-range timing dependencies throughout the entire circuit, which are encapsulated within a timing correlation matrix $\boldsymbol{C} \in \mathcal{R}^{N \times N_e}$. Here N is the total number of nodes, and N_e is the number of endpoints.

The constructed graph comprises three distinct types of nodes: input nodes, gate nodes (output bits from gate operators), and module nodes (output bits from module operators). Directed edges link each gate input bit to the gate's output bit, accurately mapping data flow and timing dependencies. On the other hand, connections to module nodes are established more selectively. Each module node connects only with those inputs of the module that directly influence it, typically those inputs of lower or equal bit positions. Taking an adder output pin at bit position m as an example, we only add directed edges from adder input pins at bit position m or lower to it. Additionally, multiplexers are treated uniquely by decomposing them into individual gates, based on the principle that a multiplexer output at bit position m is influenced only by the control signals and the input signals at the same bit position m. To facilitate backward timing propagation, reverse edges (colored grey in Fig. 3) are incorporated into the graph.

For each input node, we initialize its feature $h^{'}$ to include its input arrival time and a one-hot vector representing its type (either an input port or a constant value). For all other nodes, we include the one-hot vector denoting the operator type in their initial feature. Additionally, we consider the bit positions of a module's inputs/outputs as features, which are closely related to the intra-module delay. For each module node, we use its bit position as its node feature, and for each of its predecessor edges, we use the bit position of the corresponding predecessor node as the edge feature.

B. Forward Timing Propagation

The forward timing propagation (FTP) emulates the delay propagation process during timing analysis, propagating input arrival times (delays) from PIs to endpoints. Utilizing an asynchronous message-passing framework [31], messages propagate sequentially in topological order from inputs to endpoints, advancing level by level (as shown in Fig. 4).

Recognizing that the arrival time at a node is predominantly influenced by the maximum arrival time at its predecessors, we integrate an attention mechanism [32] to selectively highlight the most influential predecessor. For a given node v, we compute the attention coefficient for each of its predecessor nodes u, as illustrated in Equation (1). The coefficient $e_{u,v}$ quantifies the significance of predecessor node u in determining the embedding of node v.

$$e_{u,v} = \begin{cases} \sigma(\boldsymbol{a}_{\boldsymbol{m}}^{\top} W^m [h_v^{'} \mid\mid h_{u,v}^{'} \mid\mid h_u^f]), & \forall v \in \mathcal{V}_m, \\ \sigma(\boldsymbol{a}_{\boldsymbol{g}}^{\top} W^g [h_v^{'} \mid\mid h_u^f]), & \forall v \in \mathcal{V}_g. \end{cases} \quad (1)$$

Here $\mathcal{V}_m$ and $\mathcal{V}_g$ denote the set of module/gate nodes, respectively, and σ represents the LeakyReLU activation function with a negative slope of 0.2. $\boldsymbol{a}_m$ and $\boldsymbol{a}_g$ denote the learnable attention vectors, and W^m and W^g are learnable weight matrices. $\mid\mid$ denotes vector concatenation, $\cdot^{\top}$ represents transposition, $h_v^{'}$ and $h_{u,v}^{'}$ denotes the initial feature of node v and edge $(u \rightarrow v)$, respectively, and h_u^f denotes the generated embedding of node u. Different attention vectors $\boldsymbol{a}_m$ and $\boldsymbol{a}_g$ are used for module and gate nodes to address their unique characteristics.

To standardize attention coefficients across different predecessors, we apply a softmax normalization as shown below:

$$\alpha_{u,v} = \text{softmax}_v(e_{u,v}) = \frac{\exp(e_{u,v})}{\sum_{k \in \mathcal{N}(v)} \exp(e_{k,v})}, \quad (2)$$

where $\mathcal{N}(v)$ denotes the set of predecessor nodes of v. The normalized attention scores are stored on the corresponding reverse edges (grey numbers in Fig. 4) and are the basis for backward timing propagation.

The scores are then used to calculate a weighted combination of the embeddings from predecessor nodes, forming the forward timing embedding (FTE) for node v. Formally, this can be written as Equation (3).

$$h_v^f = \begin{cases} f^i(h_v^{'}), & \forall v \in \mathcal{V}_i, \\ \sigma(\sum_{u \in \mathcal{N}(v)} \alpha_{u,v} W^m [h_v^{'} \mid\mid h_{u,v}^{'} \mid\mid h_u^f]), & \forall v \in \mathcal{V}_m, \\ \sigma(\sum_{u \in \mathcal{N}(v)} \alpha_{u,v} W^g [h_v^{'} \mid\mid h_u^f]), & \forall v \in \mathcal{V}_g. \end{cases} \tag{3}$$

Here σ represents the ReLU activation function, $\mathcal{V}_i$ denotes the set of input nodes, and f^i is a multilayer perceptron model (MLP).

Driven by the above attention mechanism, FTP effectively mimics delay calculation in timing analysis. The calculated attention scores adaptively vary under different input arrival time distributions, capturing and prioritizing the most impactful predecessors.

C. Backward Timing Propagation

While FTP captures local timing correlations between nodes and their immediate predecessors, it fails to understand longer-range timing dependencies within the circuit. To address this limitation, we introduce a backward timing propagation (BTP) algorithm that evaluates the importance of each node relative to every endpoint, offering a more comprehensive, global perspective of timing dependencies within the circuit.

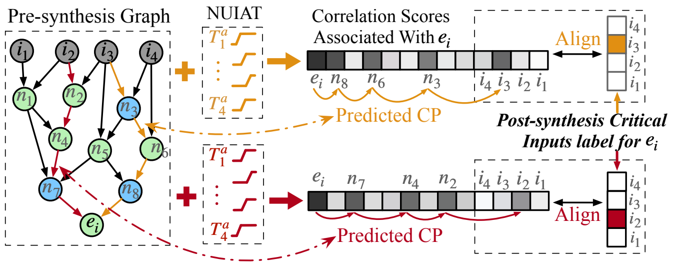
To enhance efficiency, BTP is designed to operate in batches, processing a group of endpoints simultaneously, typically with a batch size of 1024 or larger. Let N represent the total number of nodes in the graph, and N_e represent the number of endpoints in each batch. Initially, a correlation matrix $\boldsymbol{C} \in \mathcal{R}^{N \times N_e}$ is established, where the i-th row vector $\boldsymbol{C}_i \in \mathcal{R}^{N_e}$ represents the timing correlation between the i-th node v_i and the endpoints. For the endpoints themselves, their respective row correlation vectors are initialized using a one-hot embedding that uniquely identifies the index of each endpoint within the batch. The correlation vectors for all other nodes are initialized to zero.

After initialization, BTP starts from the endpoints and progresses along the reverse edges in reverse topological order. Nodes at the same reverse topological level are processed simultaneously. As shown in Fig. 4, the correlation vector of each node is updated as the weighted sum of its predecessors' correlation vectors, where the weights are given by the attention scores computed during FTP. Formally, this can be written as Equation (4), where $\mathcal{N}^r(u)$ denotes the set of predecessors of u in the reverse graph.

$$\boldsymbol{C}_i = \sum_{v_j \in \mathcal{N}^r(v_i)} \alpha_{v_i,v_j} \times \boldsymbol{C}_j. \tag{4}$$

BTP concludes when all inputs have been reached, resulting in the complete correlation matrix $\boldsymbol{C} \in \mathcal{R}^{N \times N_e}$. Each element $\boldsymbol{C}_{i,j}$ in this matrix, which ranges from 0 to 1, quantifies the probability of the i-th node being part of the critical path(s) for the j-th endpoint. Consequently, the correlation matrix effectively serves as an approximation of the critical path distribution for each endpoint. Actually, it is intuitive and straightforward to predict the critical path for an endpoint using this correlation matrix. As illustrated in Fig. 5, a greedy traversal along the reverse edges, starting from the endpoint and selecting the node with the highest correlation score at each step, can predict the critical path for a given endpoint. The process of identifying these critical paths is pivotal as it offers substantial opportunities for guiding early-stage timing optimization, which will be left for future work.

An intriguing attribute of our timing correlation matrix is described as follows:



Pre-synthesis Graph
NUIAT
Correlation Scores Associated With e_i
i_1
i_2
i_3
i_4
n_1
n_2
n_3
n_4
n_5
n_6
n_7
n_8
e_i
T_1^a
T_4^a
e_i n_8 n_6 n_3 i_4 i_3 i_2 i_1
Align
Predicted CP
Post-synthesis Critical Inputs label for e_i
e_i n_7 n_4 n_2 i_4 i_3 i_2 i_1
Align
Predicted CP

Fig. 5 An example that shows how the predicted correlation matrix aligns with the ground-truth post-synthesis critical path distribution. Here CP is for critical path, and NUIAT is for non-uniform input arrival times. Predicted critical paths for an endpoint can be obtained through reverse traversal based on the correlation scores (the darker the larger).

Property 1 (Normalized Input Correlation Scores)**.** *For any endpoint, the sum of the correlation scores for all associated inputs equals one.*

The above property emerges intuitively when we consider the backward propagation from an endpoint as a random walk along the directed reverse edges starting from that endpoint, with the edge attention scores acting as transition probabilities. The correlation score between any input and the endpoint represents the probability that the random walk terminates at that input, ensuring that the total probability for all related inputs sums to one.

The correlation matrix derived from BPN shares similarities with the self-attention mechanism observed in Graph Transformers, where importance scores are allocated to nodes in relation to one another. Building upon this understanding, we introduce a global path embedding (GPE) technique that leverages information from all nodes in the graph to capture long-range critical path details for each endpoint. The global path embedding of the j-th endpoint is computed as follows:

$$h_{e_j}^g = \sigma[\sum_{v_k \in \mathcal{V}} \frac{\boldsymbol{C}_{k,j}}{\sum_{v_k \in \mathcal{V}} \boldsymbol{C}_{k,j}} \times h_{v_k}^f \mid\mid \sum_{v_i \in \mathcal{V}_i} \boldsymbol{C}_{i,j} \times h_{v_i}^f], \tag{5}$$

where σ is the RuLU activation function, $C_{i,j}$ denotes the correlation score between the i-th node v_i and the j-th endpoint e_j, $\mathcal{V}$ and $\mathcal{V}_i$ represent the set of all nodes and all input nodes, respectively. In the above equation, we concatenate the weighted sum of all nodes' forward timing embeddings and the weighted sum of all input nodes' forward timing embeddings, where the weights represented by the correlation scores are normalized to ensure that the global path embeddings are comparable across different circuits.

Subsequently, we concatenate the forward timing embedding h^f and the global path embedding h^g to form the final embedding for each endpoint. The concatenated embedding is then processed through an MLP-based regressor, which transforms these embeddings into predicted post-synthesis arrival times. Formally, this can be written as:

$$d_j = f^p([h_{e_j}^f \mid\mid h_{e_j}^g]). \tag{6}$$

Here d_j denotes the predicted arrival time of the j-th endpoint, and f^p represents an MLP model.

D. Critical Path Aware Loss Function

While BPN effectively captures long-range timing dependencies through the correlation matrix $\boldsymbol{C} \in \mathcal{R}^{N \times N_e}$ and the global path embedding (GPE), its performance hinges on the accuracy of $\boldsymbol{C}$ in reflecting post-synthesis timing dependencies, particularly the critical path distribution. An inaccurate correlation matrix can degrade performance. To address this, a critical path aware loss (CPL) function is introduced to explicitly integrate post-synthesis critical path labels into the training process.

The primary challenge in annotating post-synthesis critical paths on the pre-synthesis circuit lies in the absence of a direct mapping between pre- and post-synthesis circuit components. To effectively address this issue, we adopt a pairwise annotation method (input-endpoint pair) that capitalizes on the consistency of input/output (IO) signals and registers between the pre- and post-synthesis designs. This consistency allows us to accurately label the critical inputs for each endpoint as the startpoints of its critical paths from the post-synthesis timing report.

Building on these annotated pairwise critical paths, we design a tailored loss function, namely critical path aware loss (CPL), that enhances the model's capability to capture long-range timing dependencies. As illustrated in Property 1, the sum of the correlation scores for all inputs related to any endpoint equals one. This indicates that the correlation score between the inputs and an endpoint can be interpreted as the probability distribution of the inputs being critical to the endpoint. Our CPL function, depicted in Fig. 5, is designed to align the model's predicted critical input distribution with the actual post-synthesis critical input labels by maximizing the correlation scores between each endpoint and its labeled critical inputs. In doing so, it also enhances the correlation scores along the critical paths that connect these critical inputs to their respective endpoints.

To preserve structural information, we opt to maximize the total correlation scores of each endpoint's near-critical inputs rather than focusing on a single input. Specifically, we target critical paths whose arrival times fall within 0.95 to 1 times the maximum arrival time of the endpoint. Near-critical paths are defined as those whose arrival times fall within 0.95 to 1 times the maximum arrival time of the endpoint. A penalty term is introduced to distribute the correlation scores more evenly among the near-critical inputs. This method ensures that we account for a broader range of critical inputs, reflecting a more comprehensive view of each endpoint's timing dependencies.

The balanced critical path score p_j for the j-th endpoint is computed as follows:

$$p_j = \sum_{v_i \in \mathcal{V}_i^c(j)} C_{i,j} - \sum_{v_i \in \mathcal{V}_i^c(j)} |C_{i,j} - \frac{1}{|V_i^c(j)|} \sum_{v_i \in \mathcal{V}_i^c(j)} C_{i,j}|, \quad (7)$$

where $\mathcal{V}_i^c(j)$ denotes the near-critical inputs for the j-th endpoint. Property 1 ensures that the critical path scores for different endpoints are comparable in magnitude.

The overall loss function is then defined as:

$$\mathcal{L} = \sum_{j=1}^{N_e} \exp(1 - p_j) \times (d_j - \hat{d_j})^2, \quad (8)$$

where d_j and $\hat{d_j}$ denote the predicted and groundtruth arrival time of the j-th endpoint, respectively. The exponential term $\exp(1 - p_j)$ serves as a weighting factor for the Mean Squared Error (MSE) loss, which is dependent on the critical path scores. By minimizing this loss function, the model not only seeks to reduce the error in arrival time predictions but also aims to maximize the total correlation scores of the post-synthesis critical inputs. This dual focus is essential for enhancing the reliability and efficacy of pre-synthesis timing predictions.

Our customized loss function offers a more explicit approach for adjusting the timing correlation matrix compared to the commonly used mean squared error (MSE) loss, which primarily supervises based on endpoint arrival time alone. Training the BPN model with this specialized loss function allows it to adaptively refine the timing correlation matrix and the on-edge attention scores. This refinement process not only enhances the forward timing embedding derived from the forward propagation but also improves the global path embedding associated with the correlation matrix. As a result, our approach leads to more precise and reliable timing predictions.

TABLE I The statistics of our synthetic dataset, which comprises 400 designs. We sample 100 sets of random input arrival time distributions for each design.

Group	#Cases	#K Nodes		# Endpoints	
		range	average	range	average
SYN1	300×100	1∼20	8	10∼150	35
SYN2	100×100	20-80	40	30∼350	150

TABLE II The detailed information about the open-source testing dataset. We sample 100 sets of random input arrival time distributions for each design.

Usage	Designs
Interface Controllers	gpio, ps2, ac97_ctrl, wb_lcd uart16550
Processing and System Control	pid_controller, systemcaes picorv32
Multimedia Processing	y_quantizer, y_huff, y_dct
Security and Signal Modulation	pwm, ecg, aes128
Memory and Interconnect	oc_wb_dma, wb_conmax, oc_mem_ctrl, wb_dma

V. Experiments

A. Experimental Setup

We developed the NUA-Timer framework with DGL [33] and PyTorch [34]. The neural networks were trained on a Linux machine with 16 Intel Xeon Gold 6226R cores (2.90GHz), one GeForce RTX 3090 Ti graphics card, and 24 GB of main memory. The embedding dimension of our BPN model is set to 128, and the MLP used for regression has 3 layers with a hidden dimension of 256. Our model was trained for 100 epochs with a learning rate of 0.001 and a batch size of 1024.

For comprehensive training, we generated a synthetic dataset of 400 RTL designs. For each design, we sampled 100 sets of random input arrival time patterns, resulting in a total of 40,000 synthetic cases. Detailed characteristics of this synthetic dataset are presented in TABLE I. We categorized the 400 designs into two groups based on design size and distributed designs in each group into training, validation, and testing subsets in a 0.7:0.1:0.2 ratio, respectively. This design-level split ensures that the test designs remain unseen during the training phase, providing an unbiased evaluation. To evaluate the generalization capability of our proposed method, we also prepared a test dataset comprising open-source designs from OpenCores [35], details of which are provided in TABLE II. The open-source dataset encompasses a broad spectrum of applications, including control, memory, multimedia, cryptography, and processor designs. Similar to the synthetic dataset, we sampled 100 sets of random input arrival time patterns for each open-source design. These open-source designs were strictly used for testing purposes to prevent any exposure during the training phase.

TABLE III Performance Comparison on the synthetic benchmark. Path-based methods suffer from the over-fitting issue, and our method outperforms all baselines on the unseen test designs.

Benchmark		ACCNN [7]		GraphGPS [12]		MasterRTL [4]		RTL-Timer [5]		NUA-Timer	
		R^2	MAPE	R^2	MAPE	R^2	MAPE	R^2	MAPE	R^2	MAPE
Train	SYN 1	0.861	4.1	-0.029	13.6	**0.917**	**3.2**	0.914	3.1	0.903	3.7
	SYN 2	0.725	4.2	-0.098	9.3	**0.906**	**2.8**	0.875	3.2	0.871	3.3
Test	SYN 1	0.843	5.1	0.009	17.1	0.72	7.6	0.853	5.5	**0.902**	**4.5**
	SYN 2	0.728	4.5	-0.127	8.2	0.556	6.2	0.715	4.9	**0.813**	**3.8**

TABLE IV Performance Comparison on the open-source test benchmark, where ep is for endpoint. All designs here are not exposed during training, and each design is sampled with 100 sets of random input arrival times. Our proposed NUT-Timer outperforms all the baselines, demonstrating its superior generalization ability.

Benchmark			ACCNN [7]		GraphGPS [12]		MasterRTL [4]		RTLtimer [5]		NUA-Timer	
Designs	#K nodes	#K ep	R^2	MAPE	R^2	MAPE	R^2	MAPE	R^2	MAPE	R^2	MAPE
gpio	1.8	0.4	0.267	14.2	-0.108	19.8	0.130	16.7	0.577	11.0	**0.901**	**5.6**
pwm	3.6	0.2	0.680	10.5	-0.414	24.9	0.765	10.5	0.764	9.6	**0.844**	**7.7**
ps2	3.7	0.2	0.553	17.0	-0.061	30.2	0.293	21.1	0.719	13.3	**0.856**	**9.8**
ac97_ctrl	7.1	0.4	0.640	14.7	0.000	29.0	0.020	24.5	0.503	13.7	**0.839**	**8.4**
pid_controller	9.0	0.4	0.471	11.2	-0.700	23.1	0.505	12.5	0.784	7.9	**0.827**	**6.7**
wb_lcd	10.1	0.6	0.653	9.4	-0.262	21.4	-0.010	19.4	0.680	9.7	**0.914**	**5.0**
uart16550	11.3	0.6	0.390	9.0	-0.737	19.0	0.032	14.4	0.296	10.9	**0.859**	**4.9**
systemcaes	15.4	0.7	0.234	12.6	-1.181	28.6	0.620	11.5	**0.748**	**8.7**	0.682	9.9
picorv32	28.6	1.6	0.736	6.3	-2.147	22.5	0.367	10.1	0.485	9.2	**0.746**	**6.0**
wb_dma	32.5	0.7	0.115	21.8	-0.405	31.7	0.717	12.5	0.670	11.4	**0.916**	**5.9**
oc_wb_dma	32.5	2.0	0.046	11.6	-1.467	28.9	0.649	11.1	0.712	8.6	**0.911**	**4.9**
oc_mem_ctrl	34.9	1.8	-0.718	24.1	-1.117	34.4	0.513	18.2	0.413	23.2	**0.718**	**12.0**
y_quantizer	40.7	2.8	0.585	18.0	-0.036	32.8	0.227	19.3	-0.435	27.4	**0.656**	**14.8**
y_huff	41.6	2.4	0.578	13.4	-0.009	22.1	-0.206	21.8	-0.307	20.2	**0.785**	**9.2**
wb_conmax	59.8	2.0	0.239	11.1	-1.489	21.3	0.557	9.8	0.838	5.4	**0.847**	**4.8**
ecg	177.7	7.3	-0.001	13.8	-0.057	21.4	0.182	16.5	0.081	16.9	**0.831**	**6.4**
y_dct	227.6	5.3	-0.256	15.3	-0.406	23.0	0.694	12.0	0.497	17.0	**0.732**	**9.3**
aes128	256.5	10.7	0.518	22.1	-0.137	43.9	0.528	16.8	0.114	25.1	**0.921**	**6.8**
AVG.	55.2	2.2	0.318	14.2	-0.596	26.6	0.366	15.5	0.452	13.8	**0.821**	**7.67**

For experiments, we evaluate our proposed method within the FPGA design flow, where the circuit's pre-synthesis and post-synthesis topologies differ significantly, posing substantial challenges for accurate pre-synthesis timing prediction. The task is to predict the post-synthesis arrival times at the timing endpoints based on only pre-synthesis information. To collect ground-truth labels, we use a commercial tool for logic synthesis and post-synthesis timing analysis. The FPGA board employed for these experiments consists of over 400,000 Look-Up Tables (LUTs) and 400,000 D-type Flip-Flops (DFFs), supporting a DDR memory interface with a data rate of 1866 Mbps and high-speed serial transceivers (SerDes) operating at 12.5 Gbps. The device is packaged in a Fine-pitch Ball Grid Array (FCBGA) with 900 balls (FCBGA900).

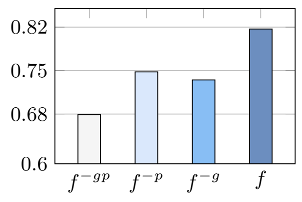
We compare NUA-Timer against state-of-the-art pre-synthesis timing evaluation methods, including path-based methods [4], [5] and graph-based methods [7], [12]. MasterRTL [4] identifies a single pre-synthesis critical path for each endpoint and applies a tree-based model (e.g., Random Forest) with feature engineering to make predictions. Similarly, RTL-Timer [5] relies on feature engineering and tree-based models but adds randomly sampled paths along with the critical path for each endpoint. The endpoint arrival times are then calculated based on the maximum predicted path delays. On the other hand, the graph-based approach ACCNN [7] estimates the endpoint arrival time by analyzing the fanin structures and topologies, encoded through a GNN. Additionally, we include the Graph Transformer (GT) [12] as a baseline model, renowned for its efficacy in capturing long-range dependencies within graphs. To ensure a fair comparison, we have re-implemented the baseline methods to align with our specific problem context and have employed the official hyperparameters during the training phase.

To evaluate the model's performance, we employ two metrics: the coefficient of determination (R^2) and the mean absolute percentage error (MAPE). The R^2 metric is crucial as it measures the proportion of variance in the dependent variable that is predictable from the independent variables, effectively assessing how well the model fits the data. Conversely, MAPE offers insight into the model's accuracy by providing a direct average of the percentage errors made by the model's predictions. These metrics together give a comprehensive view of the model's predictive capabilities and accuracy.

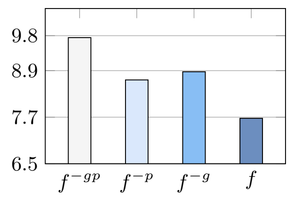
B. Overall Performance

As mentioned above, all the methods were trained using a synthetic dataset and subsequently evaluated on both unseen synthetic and open-source datasets. As indicated by the results listed in TABLE III and TABLE IV, our proposed method NUA-Timer outperforms the baseline methods on both synthetic and open-source testing designs. Particularly, NUA-Timer shows a notably larger improvement compared to baseline methods on the real-world open-source designs. The reason may be that the open-source designs share less similarity with the training synthetic designs, necessitating a strong generalization ability to perform well on these designs.

From the results listed in TABLE III, we find that the path-based methods [4], [5] suffer from the over-fitting issue, showing impressive results on training datasets but failing to generalize effectively to new,



(a) Average R^2



(b) Average MAPE (%)

Fig. 6 Results of ablation study, measured by the average performance on the open-source benchmark.

unseen data. This limitation stems from variations in post-synthesis critical path distributions across different designs, which can significantly differ from the pre-synthesis critical paths used for predictions by these methods. On the other hand, the GNN-based method [7] exhibits less tendency to overfit on the synthetic dataset but falls short in performance due to its localized message aggregation scheme. This method struggles to capture long-range timing dependencies, which are essential for accurate timing estimation under varying input arrival times. Additionally, the graph transformer-based baseline [12] does not yield satisfactory results in timing prediction, primarily due to its inability to adequately understand critical timing paths.

To comprehensively evaluate the generalization capabilities of the models, experiments were also conducted on the benchmark consisting of open-source designs. The results listed in TABLE IV indicate a pronounced decline in performance by the baseline methods when applied to unseen real-world designs, particularly with larger designs. This performance dip may be attributed to the lower similarity between the open-source designs and the training synthetic designs, which increases the prediction difficulty. This underscores the insufficiency of manual feature engineering or simplistic graph neural networks in accurately capturing long-range timing dependencies under non-uniform input arrival times. Our proposed NUA-Timer achieves a substantial reduction in MAPE by 44.2% and an enhancement in R^2 by 82.2% on average when compared to the best-performing baseline [5]. Notably, our method excels even with testing designs significantly larger than those used in training, showcasing its superior scalability and generalization capabilities. This robustness and adaptability underscore the effectiveness of our approach in handling diverse and complex design challenges in real-world scenarios.

In summary, our proposed approach demonstrates a robust capability to generalize and accurately predict timing across a diverse range of circuit designs, thanks to its comprehensive handling of intricate timing dependencies.

C. Ablation Study

To further assess the efficacy of the proposed techniques, we conducted ablation studies. Let p denote our critical path aware loss (CPL) function, g denote the global path embedding (GPE), f represent the full version of our method, and f^{-x} represent the model without component x. Specifically, we compare the performance of four models: f^{-gp}, f^{-g}, f^{-p} and f, on the open-source testing benchmark. As shown in Fig. 6, the removal of either CPL or GPE leads to suboptimal performance, while removing both of them leads to the worst performance. Quantitatively, removing CPL results in a reduction of the average R^2 from 0.82 to 0.75 and an increase in average MAPE from 7.7% to 8.7%. Removing GPE results in an even more significant drop, with the average R^2 decreasing to 0.73 and the average MAPE increasing to 8.9%. The combined removal

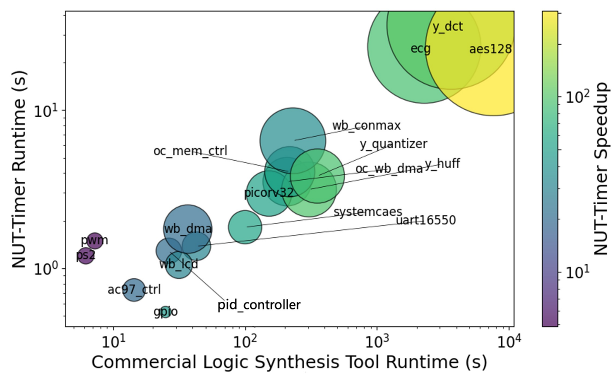


Fig. 7 NUA-Timer runtime vs. commercial tool runtime.

of both components results in an R^2 of 0.68 and an MAPE of 9.8%. These results demonstrate the significant contribution of both CPL and GPE to the overall performance of our method.

D. Runtime Analysis

Fig. 7 shows the runtime comparison between NUA-Timer and the commercial tool to obtain post-synthesis timing evaluation. The runtime for NUA-Timer includes both preprocessing and model inference, whereas the runtime for the baseline tool includes logic synthesis and timing analysis. We calculate the average runtime across 100 different sets of random input arrival times. These evaluations are conducted on the same CPU machine to maintain consistency in the comparison.

Each design in the analysis is denoted by a circle, where the size of the circle is proportional to the size of the design. The variation in the color shades indicates the relative speedup of NUA-Timer compared to the baseline, with darker shades representing slower speeds and lighter shades indicating faster speeds. Notably, NUA-Timer demonstrates a significant increase in speedup for larger designs. For instance, it achieves an impressive speedup exceeding 300 times for the aes128 design. On average, NUA-Timer facilitates a speedup of over 60 times, highlighting its superior efficiency.

VI. Conclusion

This paper introduces NUA-Timer, a timing prediction framework addressing non-uniform input arrival times in hierarchical designs. We develop a Bidirectional Propagation Neural Network (BPN) to capture long-range timing dependencies and accurately reflect post-synthesis critical paths. The BPN model begins with an attention-based forward stage to capture local timing dependencies, following a backward propagation process to capture the long-range timing dependencies across the whole circuit. A customized loss function, informed by post-synthesis critical path information, is utilized to enhance the training process of our model. Experiments on both synthetic and open-source designs demonstrate NUA-Timer's superiority over state-of-the-art methods. It is important to note that our proposed methodology is universal and can be applied broadly across various timing prediction tasks. Looking ahead, our future work will aim to apply our method to other circuit formats such as ASICs and other EDA stages like physical design. We are also interested in integrating our method into logic synthesis flows, e.g., guiding timing optimization based on predicted critical paths.

Acknowledgements

The project is supported in part by Research Grants Council of Hong Kong SAR (No. RFS2425-4S02 and No. CUHK14211824).

References

[1] C. Xu, C. Kjellqvist, and L. W. Wills, "SNS's not a synthesizer: a deep-learning-based synthesis predictor," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2022.

[2] K.-M. Lai, T.-W. Huang, P.-Y. Lee, and T.-Y. Ho, "ATM: A high accuracy extracted timing model for hierarchical timing analysis," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2021.

[3] B. Li, N. Chen, M. Schmidt, W. Schneider, and U. Schlichtmann, "On hierarchical statistical static timing analysis," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2009.

[4] W. Fang, Y. Lu, S. Liu, Q. Zhang, C. Xu, L. W. Wills, H. Zhang, and Z. Xie, "MasterRTL: A pre-synthesis PPA estimation framework for any RTL design," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023.

[5] W. Fang, S. Liu, H. Zhang, and Z. Xie, "Annotating slack directly on your verilog: Fine-grained rtl timing evaluation for early optimization," in *ACM/IEEE Design Automation Conference (DAC)*, 2024.

[6] Y. Ouyang, S. Li, D. Zuo, H. Fan, and Y. Ma, "ASAP: Accurate Synthesis Analysis and Prediction with Multi-Task Learning," in *ACM/IEEE Workshop on Machine Learning CAD (MLCAD)*, 2023.

[7] H. Zheng, Z. He, F. Liu, Z. Pei, and B. Yu, "LSTP: A Logic Synthesis Timing Predictor," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2024.

[8] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[9] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?" in *International Conference on Learning Representations (ICLR)*, 2018.

[10] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *The Web Conference*, 2020.

[11] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 28 877–28 888, 2021.

[12] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a general, powerful, scalable graph transformer," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 14 501–14 515, 2022.

[13] P.-Y. Lee and I. H.-R. Jiang, "iTimerM: A compact and accurate timing macro model for efficient hierarchical timing analysis," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 4, pp. 1–21, 2018.

[14] F. Dartu and Q. Wu, "To do or not to do hierarchical timing?" in *ACM International Symposium on Physical Design (ISPD)*, 2013.

[15] T.-Y. Lai, T.-W. Huang, and M. D. Wong, "LibAbs: An efficient and accurate timing macro-modeling algorithm for large hierarchical designs," in *ACM/IEEE Design Automation Conference (DAC)*, 2017.

[16] K. K.-C. Chang, C.-Y. Chiang, P.-Y. Lee, and I. H.-R. Jiang, "Timing macro modeling with graph neural networks," in *ACM/IEEE Design Automation Conference (DAC)*, 2022.

[17] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *ACM/IEEE Design Automation Conference (DAC)*, 2019.

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

[19] X. He, Z. Fu, Y. Wang, C. Liu, and Y. Guo, "Accurate timing prediction at placement stage with look-ahead RC network," in *ACM/IEEE Design Automation Conference (DAC)*, 2022.

[20] P. Cao, G. He, and T. Yang, "Tf-predictor: Transformer-based prerouting path delay prediction framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 7, pp. 2227–2237, 2022.

[21] Z. Xie, R. Liang, X. Xu, J. Hu, C.-C. Chang, J. Pan, and Y. Chen, "Preplacement net length and timing estimation by customized graph neural network," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 11, pp. 4667–4680, 2022.

[22] Q. Song, X. Cheng, and P. Cao, "Critical paths prediction under multiple corners based on bilstm network," in *ACM/IEEE Design Automation Conference (DAC)*, 2023.

[23] 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 *ACM/IEEE Design Automation Conference (DAC)*, 2022.

[24] G. He, W. Ding, Y. Ye, X. Cheng, Q. Song, and P. Cao, "An optimization-aware pre-routing timing prediction framework based on heterogeneous graph learning," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2024.

[25] Z. Wang, S. Liu, Y. Pu, S. Chen, T.-Y. Ho, and B. Yu, "Restructure-tolerant timing prediction via multimodal fusion," in *ACM/IEEE Design Automation Conference (DAC)*, 2023.

[26] X. Peng, X. Song, and Y. Zou, "PreRoutSGAT: Pre-Routing Timing Prediction Based on Graph Neural Network with Global Attention," 2024.

[27] 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.

[28] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *ACM/IEEE Design Automation Conference (DAC)*, 2022.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[30] N. Keriven and S. Vaiter, "What functions can graph neural networks compute on random graphs? the role of positional encoding," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 36, pp. 11 823–11 849, 2023.

[31] Z. He, Z. Wang, C. Bai, H. Yang, and B. Yu, "Graph learning-based arithmetic block identification," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2021.

[32] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *International Conference on Learning Representations (ICLR)*, 2018.

[33] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," *arXiv preprint arXiv:1909.01315*, 2019.

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[35] "OpenCores," https://opencores.org.