Timing-Driven Technology Mapping Approximation Based on Reinforcement Learning

Yuyang Ye, Tinghuan Chen, *Member, IEEE*, Yifei Gao, Hao Yan, *Member, IEEE*, Bei Yu, *Senior Member, IEEE*, and Longxing Shi, *Senior Member, IEEE*

***Abstract*—As the transistor technology nodes shrink into the nano-scales, timing guardbands caused by aging effects and process variations continue to increase. Approximate computing can eliminate aging-and-variation-induced timing guardbands without sacrificing the design performance. It can apply local approximate changes (LACs) automatically in circuits to reduce critical path delay (CPD). However, efficiently achieving timing optimization under error distance constraints is still tricky. This work proposes an automated timing-driven technology mapping approximation framework based on reinforcement learning (RL). The framework uses path-weighted graph neural networks (PGNNs) to embed RL states and timing path-aware LAC candidates to construct RL action spaces. It can efficiently eliminate timing guardbands induced by aging and variation. Our proposed circuit-agnostic framework operates on the gate-level netlists. According to experiments on the open-source circuits using TSMC 28 and 16-nm technology under aging and variation conditions, our framework can achieve an average 24.78% CPD reduction under five different error distance constraints and 4.83× speedup, compared with a state-of-the-art method.**

***Index Terms*—Approximate computing, approximate logic synthesis, timing optimization.**

I. Introduction

With the continuous shrinking of CMOS technology nodes, transistor aging effects, such as negative bias temperature instability (NBTI), and process variations make timing closure and signoff increasingly challenging [1]. To guarantee the parametric yield and circuit lifetime, designers add performance-degrading timing guardbands on the operational clock period. However, with the technology scaling down, these aging-and-variation-induced timing guardbands are increasing rapidly, eventually resulting in severe timing performance degradation [2]. Therefore, there is an urgent need for effective methodologies in timing optimization that can mitigate the impacts of aging and variation without compromising performance.

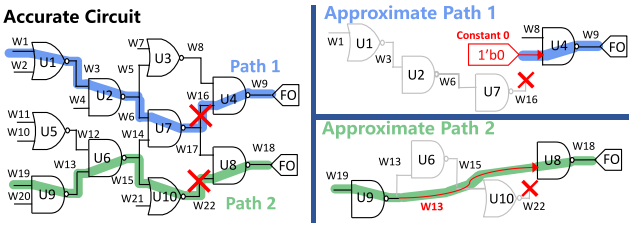

Fig. 1. Descriptive example of achieving timing optimization by approximate computing. Left: The accurate circuit; top right: the approximate path applying wire-by-constant LAC; and bottom right: the approximate path applying wire-by-wire LAC.

Approximate computing has emerged as a promising design paradigm to improve timing performance. It introduces slight computational imprecision to reduce circuit delay and has proven effective in numerous error-tolerant applications [3]. Prior efforts primarily focused on approximate circuit design, often relying on human expertise. Recently, some studies have proposed methods for automating approximate computing within functional error constraints [3], [4], [5], [6], [7], [8], [9]. Fig. 1 shows an example, where approximate computing can reduce path delay by modifying local structures in circuits. Such modifications are called local approximate changes (LACs).

Several previous works [3], [4], [5], [6], [7] have approached approximate logic optimization to enhance timing, using methods like the greedy algorithm. PowerX [6] utilizes deep learning to accelerate the flow by predicting logic errors. SEALS [5] consider the relationship between gate and output, termed "sensitivity," to achieve speedups. HEDALS [4] focuses on achieving approximate computing on critical timing paths. Since they work on circuits with AND-Inverter graph representations rather than real technology-dependent circuits, the timing information is unrealistic at the Boolean level. Similarly, technology mapping approximation efforts [8], [9], utilizing genetic algorithms, aim to optimize timing in mapped circuits, factoring in aging effects and process variations for a reliability-aware design.

Error metrics are crucial in evaluating the accuracy of approximate circuits. Two common error metrics are error rate (ER) and distance. The ER is important for random/control circuits, and error distance is more critical for arithmetic circuits.

Manuscript received 25 July 2023; revised 2 November 2023 and 5 February 2024; accepted 6 March 2024. Date of publication 20 March 2024; date of current version 22 August 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFB4402900; in part by the National Natural Science Foundation of China under Grant 62274034, Grant 62304197, and Grant 61904030; and in part by the Research Grants Council of Hong Kong SAR under Project CUHK14210723. This article was recommended by Associate Editor H. Zheng. *(Corresponding authors: Hao Yan; Tinghuan Chen.)*

Yuyang Ye, Yifei Gao, Hao Yan, and Longxing Shi are with the National ASIC Research Center, Southeast University, Nanjing 210096, China, and also with National Center of Technology Innovation for EDA, Nanjing 210096, China (e-mail: yanhao@seu.edu.cn; chentinghuan@cuhk.edu.cn).

Tinghuan Chen is with the School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen 518172, China.

Bei Yu is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR.

Digital Object Identifier 10.1109/TCAD.2024.3379016

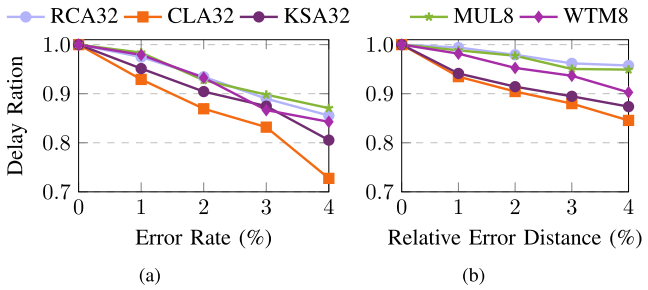

Fig. 2. Delay ratios of the approximate circuits obtained by VECBEE-SASIMI over the accurate circuits for different designs under (a) different ER constraints and (b) different error distance constraints.

Recent studies in approximate computing have successfully achieved timing optimization under ER constraints. As shown in Fig. 2(a), VECBEE-SASIMI [3] reduces critical path delay (CPD) under different ER constraints. The most critical paths in arithmetic circuits are timing paths to the most significant bits (MSBs) [10], where the signal value changes on MSBs always result in significant error distances in previous works. As demonstrated in Fig. 2(b), it causes the poor optimization performance of VECBEE-SASIMI. Thus, the challenge lies in optimizing timing under varying error distance constraints without comprising circuit accuracy.

In summary, three main reasons cause poor timing optimization under error distance constraints in previous works: 1) various pseudo-linear heuristics or analytical methods, including genetic algorithms and greedy methods, result in suboptimal solutions; 2) previous works treat the process as a black-box optimization problem, lacking accurate timing path modeling, which causes low efficiency; and 3) the consideration of the relationship between timing paths and LACs is limited in other proposed works, which makes it challenging to select LACs that can optimize timing performance under specific error distance constraints without extensive simulations.

Reinforcement learning (RL) has emerged as a powerful machine learning paradigm, demonstrating superhuman performance in various optimization tasks [11]. In RL, an agent learns an optimal policy through trial and error in an interactive environment to maximize rewards. This involves understanding the state of the environment, actions to alter it, and rewards as feedback. Recent applications of RL in electronic design automation, including gate sizing, transistor sizing [12], physical design placement [13], and technology mapping [14], have shown promising results. In this article, we customize RL for technology mapping approximation to optimize timing performance under error distance constraints.

For RL state embedding, the timing path cannot be accurately modeled while collecting circuit features. However, the circuit features on timing paths, especially critical paths, are important in timing optimization. The essential reason is that the equipped graph neural networks (GNNs) in previous frameworks can only learn circuit information from part of gates on timing paths due to over-smoothing issue [11], [12], [13]. The RL action spaces significantly affect the RL agent's efficiency, where too vast action space causes low-convergence speed and too narrow action space causes terrible optimization quality [12]. Following previous approximate logic optimization and technology mapping approximation works [3], [4], [7], [8], [9], the action spaces can be narrowed by generating LAC candidates. However, these candidates are identified without considering relationships between timing paths and LACs. It only optimizes timing under ER constraints, thus causing poor timing optimization performance. Thus, the RL action spaces for our framework cannot be defined reasonably by LAC candidates without considering the RL agent application.

In this work, we propose a timing-driven technology mapping approximation framework based on RL. The framework intelligently achieves approximate computing under predictable error distance constraints. It reduces CPD considering aging effects and process variations. Our RL agent has path-weighted GNNs (PGNNs) and timing path-aware LAC candidates. For embedding the RL state, the PGNNs open the black box of technology mapping approximation through learning circuit information on global timing paths without over-smoothing issues. For constructing a reasonable action space, the timing path-aware LAC candidates can not only narrow the action space but achieve timing optimization. The customized RL states and actions can help the proposed RL agent achieve higher-optimization efficiency and more powerful model generalizability. According to aging and variation-aware experiments on the TSMC 28 and 16-nm technology, our framework can achieve an average of 24.78% CPD reduction under 5 different error distance constraints. The timing optimization takes 70.36 mins on average for arithmetic circuits ranging from 2k to 30k gates. We highlight our contributions in detail.

1) We present a novel timing-driven technology mapping approximation framework based on RL at the gate level. It can perform approximate computing to optimize aging-and-variation-aware timing performance under error distance constraints automatically and efficiently.
2) We propose PGNNs, namely, PGNNs, to model timing paths accurately by learning global information on timing paths and local information in neighborhoods, which helps embed RL states.
3) We generate timing path-aware LAC candidates considering the relationships between timing paths and LACs, which helps construct RL action spaces. The generated candidates can optimize circuit timing performance under error distance constraints.
4) Our circuit-agnostic framework is evaluated with open-source designs. The results demonstrate that our framework can efficiently achieve significant optimization performance on unseen circuits with different functions and sizes. Also, our framework can work while meeting different function accuracy requirements.

II. Preliminaries

A. Local Approximate Changes

Fig. 1 presents examples of accurate circuit and two state-of-the-art LACs, including *wire-by-constant* [15] and *wire-by-wire* [16] replacements. These LACs have been

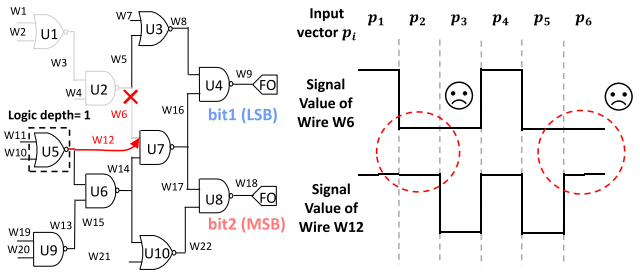

Fig. 3. Example of similarity score and logic depth. Wire W6 is the target wire and wire W12 is the LAC candidate. Under different input vectors p_i, wire W6, and W12 signal values are shown on the right. They should be the same under as many input vectors as possible to meet ER requirements.

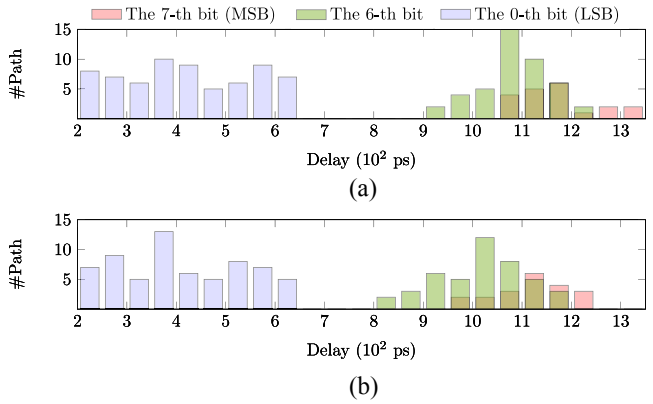

Fig. 4. Delay distribution of the timing paths to the seventh bit (red), sixth bit (green) and zeroth bit (blue) for an accurate (a) and approximate (b) 8-bit adder.

extensively used in design methodologies to improve design performance through approximate computing. Also, our framework can be extended to include other local approximation changes.

Wire-by-constant substitutes a wire in the circuit with a constant logic value "1" or "0". The top right part of Fig. 1 shows an example of wire-by-constant approximation. The replacement of wire W16 by the constant '0' is demonstrated. Thus, wire W16 is the target wire and the replacement of constant 0 is the selected LAC. Paths that previously comprised W16 are significantly accelerated by omitting the propagation delay of gates U1, U2 and U7. Wire-by-wire replaces a wire (approximated wire) in the circuit with another wire (approximation wire). The bottom right part of Fig. 1 depicts an example of wire-by-wire approximation, where W13 replaces W22. Wire W22 is the target wire, and wire W13 is the selected LAC replacement. Similarly, paths that previously comprised W22 are accelerated by omitting the propagation delay of gates U6 and U10. Thus, all the delays of timing paths comprising these wires become potentially more minor under negligible and predictable logic errors.

For large circuits, the action space becomes vast, making the optimal solution difficult to find within reasonable time and resource limits. Previous works [3], [4], [5], [6], [7], [8], [9] narrow the action space by generating the LAC candidates based on *similarity scores* and *logic depths*. The similarity score represents the percentage of cycles when the target wire and the LAC hold the same value. It can be obtained through logic simulation. The higher the value, the lower the ER. The logic depth represents the propagation depth from inputs to the LAC, which can be obtained by parsing circuits.

Examples: As shown in Fig. 3, we give an example to compute similarity score and logic depth. For wire W6, the similarity score of its ALC candidate W12 ($S^{\mathrm{W6}}_{\mathrm{W12}}$) is computed as

$$S^{\mathrm{W6}}_{\mathrm{W12}} = \sum_{p_i}^{N_{p_i}} \frac{(\mathrm{W12}_{p_i} == \mathrm{W6}_{p_i})?}{N_{p_i}} \tag{1}$$

where "$(\mathrm{W12}_{p_i} == \mathrm{W6}_{p_i})$?" equals to 1 if the signal value of W12 is the same with W6 under input vector p_i. Otherwise, it equals to 0. N_{p_i} is the input vector number. And the logic depth of W12 is 1.

B. Critical Paths Versus Timing Paths to High Bits

It is necessary to wait for the computational results of low bits when computing the results of high bits. In the accurate arithmetic circuits, the most critical paths are timing paths to the MSBs [10]. An example is illustrated in Fig. 4, which shows the delay distribution for different bit outputs in an 8-bit adder. It is observed that the delay for the seventh bit (MSB) is notably higher compared to the Sub-MSB and the least significant bit (LSB) in accurate circuits. After approximating, the delays of timing paths to the MSBs are reduced. Then, timing paths to the higher bits have a higher probability of being new critical paths.

C. Timing and Logic Error Evaluator

Our work requires accurate and fast timing and logic error evaluator to improve optimization performance. This evaluator consists of two parts:

Timing Evaluation: Aging effects and process variations influence the timing performance, which can be analyzed through aging- and variation-aware static timing analysis (STA). Compared with classical STA flow, the extra gate-level aging model and variation timing libraries are employed. For aging effects, we follow the flow [17] to generate aging-aware timing libraries based on SPICE Monte-Carlo simulation with MOSRA aging model [18]. For process variations, we adopt parametric on-chip variation analysis [19]. Both aging-aware and variation-aware timing libraries are generated using Synopsys PrimeLib [20]. Then STA is performed with Synopsys PrimeTime. It outputs CPD for the approximate circuit, which is defined as follows.

Definition 1 [CPD]: The largest path delay includes additional aging-and-variation-induced timing guardbands. It is also the minimum value of the specified clock period.

The critical path delay CPD can be computed by (2). $\mathcal{D}$ is the delay distribution of the most critical path considering aging and variation effects. $\mu(\mathcal{D})$ and $\sigma(\mathcal{D})$ represent mean and standard deviation values of delay distribution $\mathcal{D}$, respectively. CPD can cover typical aging-and-variation-induced timing guardbands [1]. Thus, it is used to evaluate the

reliability-aware timing optimization performance

$$\text{CPD} = \mu(\mathcal{D}) + 3 \times \sigma(\mathcal{D}). \tag{2}$$

Logic Error Evaluation: The traditional simulation-based method is time-consuming. In this work, we utilize the method [3], namely, VECBEE, to efficiently achieve logic error evaluation for approximate circuits. It is based on Monte Carlo simulation and an efficient technique to capture a signal change due to approximation that will cause a logic error. It outputs error distance and ER for the approximate circuit, which is defined as follows.

Definition 2 (Error Distance): The difference between the approximate and accurate circuit output values under one input vector.

Definition 3 (ER): The percentage of input vectors that the approximate circuit output differs from the exact circuit.

Normalized mean error distance (NMED) and mean relative error distance (MRED) are the mean error distance normalized by the maximum output value and the corresponding output value. They can be computed by (3) and (4), where $Y_{p_i}^{\text{acc}}$ and $Y_{p_i}^{\text{app}}$ are the circuit output values of the accurate circuit and approximate circuit under input vector p_i, respectively. N_{p_i} is the number of input vectors for testing and MSB is the value of MSB. For example, MSB equals 63 for a 64-bit unsigned adder

$$\text{NMED} = \sum_{p_i}^{N_{p_i}} \frac{\left| Y_{p_i}^{\text{acc}} - Y_{p_i}^{\text{app}} \right|}{N_{p_i} \times \left(2^{\text{MSB}+1} - 1\right)} \tag{3}$$

$$\text{MRED} = \sum_{p_i}^{N_{p_i}} \frac{\left| Y_{p_i}^{\text{acc}} - Y_{p_i}^{\text{app}} \right|}{N_{p_i} \times Y_{p_i}^{\text{acc}}}. \tag{4}$$

ERs can be computed by (5), where $O_{p_i}^{\text{acc}}$ and $O_{p_i}^{\text{app}}$ are the circuit output bits of the accurate circuit and approximate circuit under input vector p_i, respectively

$$\text{ER} = \sum_{p_i}^{N_{p_i}} \frac{O_{p_i}^{\text{acc}} \neq O_{p_i}^{\text{app}}}{N_{p_i}}. \tag{5}$$

D. Problem Formulation

Our essential parts contain circuit timing and logic information. Timing information helps improve timing performance and logic information helps meet the error distance constraint. We define timing wire as follows.

Definition 4 (Timing Wire): A timing wire includes aging-and-variation-aware timing information and logic information in the driving gate and loading gate.

An example of timing wire is illustrated in Fig. 1, and the details of information in timing wire are shown in Table I.

Based on these definitions, the technology mapping approximation problem can be formulated as a timing optimization problem under error distance as follows.

Problem 1 (Technology Mapping Approximation): During technology mapping, given a mapped accurate circuit with aging-and-variation-aware timing information and logic information, determine the optimal LACs for timing wires to generate approximate circuits with maximum CPD reductions under error constraints.

TABLE I
TIMING WIRE FEATURES USED IN OUR WORK

Type	Name	Description
Timing	wst arrive time	propagation delay to the timing wire
	dri. of dri.	drive strength of driving gate
	dri. of load.	drive strength of loading gate
	μ(delay) of dri.	mean value of driving gate's delay
	μ(delay) of load.	mean value of loading gate's delay
	σ(delay) of dri.	deviation value of driving gate's delay
	σ(delay) of load.	deviation value of loading gate's delay
	cap. of dri.	capacitance of driving gate
	cap. of load.	capacitance of loading gate
	wst input slew of dri.	worst transition time of driving gate
	wst input slew of load.	worst transition time of loading gate
	thre.voltage of dri.	threshold voltage of driving gate
	thre.voltage of load.	threshold voltage of loading gate
Logic	func. of dri.	functionality of driving gate
	func. of load.	functionality of load gate
	in. signal pro. of dri.	input signal probability of driving gate
	in. signal pro. of load.	input signal probability of loading gate

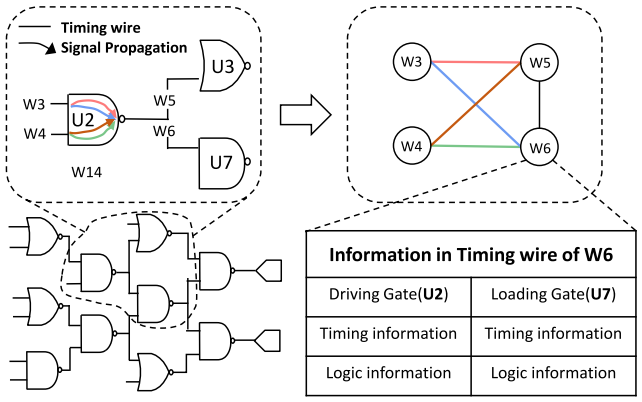

Fig. 5. Example of node and edge representation. Timing wire W6 contains the timing and logic information of its driving gate (U2) and loading gate (U7).

III. OUR PROPOSED FRAMEWORK

A. Data Preparation

In this article, we apply graph learning in an optimization loop. The accurate circuit is translated into a graph $\mathbb{G} = (\mathcal{V}, \mathcal{E})$ consisting of a node set $\mathcal{V}$ and an edge set $\mathcal{E}$. Fig. 5 gives an example of node and edge representation, where nodes are timing wires and edges are signal propagation relationships between two single timing wires. The circuit graph $\mathbb{G}$ is represented with node feature matrix X, adjacency matrix J. Node feature matrix $X : \{x_k, k \in \mathcal{V}\}$ is composed of feature vectors for all timing wires (nodes). They include their driving and loading gates' aging-and-variation-aware timing and logic information. The details of features in x_k for timing wire (node) k are shown in Table I. These features are carefully chosen based on domain knowledge after many experiments. They are collected via aging-and-variation-aware STA and logic simulation. They are expected to characterize the impact of timing wires on circuit timing performance and function accuracy under aging effects and process variations.

B. RL State

RL states $S : \{s_i, i \in \mathcal{V}_s\}$ represent selected timing wires after embedding with circuit information through PGNNs. The

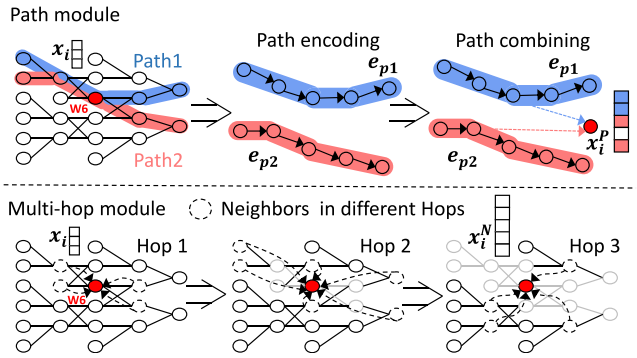

Fig. 6. Illustration of PGNNs with path and multihop module.

progress of generating RL states can be divided into selection and embedding.

In the selection stage, we generate a selected timing wire set $\mathcal{V}_s \in \mathcal{V}$. It includes independent timing wires on potential critical paths (worst-100 timing paths to each output). They can be generated through aging-and-variation-aware STA, introduced in Section II-C. In addition, the RL agent applies multiple LACs simultaneously in each round to speed up the optimization flow. However, the mutual influence of multiple LACs can affect the circuit timing and error performance [21]. Thus, we follow the method in [21] to ensure the selected timing wires on potential critical paths are almost independent. Then, we can obtain a selected timing wire set $\mathcal{V}_s$.

In the embedding stage, we propose PGNNs to embed circuit information on timing paths and local neighborhoods from the features of selected timing wires $X : \{\vec{x}_i, i \in \mathcal{V}_s\}$, and generate embedded RL states $S : \{s_i, i \in \mathcal{V}_s\}$. The information embedded in the RL state is proven useful while improving the RL agent generalizability for solving different circuit topologies and the quality of optimization. We give more details about PGNNs as follows.

PGNNs are composed of two modules. A path module can capture timing and logic information on global timing paths and a multihop module can capture this information in the local neighborhood. The timing paths can be modeled accurately in PGNNs, which is still unsolved in other EDA works [11], [12], [13].

Path Module: In the path module, the timing and logic information of the worst-10 timing wire paths are aggregated to generate path-based timing wire representations $X^P : \{x_i^P, i \in \mathcal{V}_s\}$. As shown in Fig. 6, the new timing wire representation for timing wire W6 is generated through path encoding and path combining. As an example, we introduce the generating path-based timing wire representation x_i^P for timing wire i through learning information on timing paths.

First, each path p in the path set $\mathcal{N}_i^P$ (the worst-10 timing paths through timing wire i) is encoded. It is achieved through learning the structural and semantic information of all nodes $\{x_{j_p}, j_p \in p\}$ to generate path embeddings $\{e_p, p \in \mathcal{N}_i^P\}$. Unlike classical GNNs in an unordered way, a path naturally comes with an order, preserving the ordered connections among the nodes on the path. Thus, this step aims to learn information about all the nodes on the path p and consider the order of nodes on the path, preserving more relational information among the connected nodes. To achieve path encoding, we choose *a simple sequence encoder* gate recurrent unit (GRU) [22] as the message function. For the path $p \in \mathcal{N}_i^p$, we can get the path embedding e_p followed as (6), where θ^g represents all the learnable parameters

$$e_p = \mathrm{GRU}_{\theta^g}\big(\{x_{j_p}, j_p \in p\}\big). \tag{6}$$

After encoding all paths in the path set $\mathcal{N}_i^P$, we need to combine these path embeddings $\{e_p, p \in \mathcal{N}_i^P\}$ together. Different paths have different contributions to the target timing wire representation. Thus, we adopt *an attention mechanism* α_p to learn the different weights

$$\alpha_p = \frac{\exp\big(\mathrm{LeakyReLU}(\theta^a \cdot e_p)\big)}{\sum_{p_k \in \mathcal{N}_i^p} \exp\big(\mathrm{LeakyReLU}(\theta^a \cdot e_{p_k})\big)}. \tag{7}$$

Based on the attention mechanisms, the path-based timing wire representation for timing wire i is expressed as

$$x_i^P = \sigma\left(\sum_{p \in \mathcal{N}_i^p} \alpha_p \cdot e_p\right). \tag{8}$$

Multihop Module: The multihop module is employed to learn graph-structured information in local neighborhoods. In the multihop module, the timing and logic information of neighbor timing wires is aggregated to generate neighbor-based timing wire representations $X^N = \{x_i^N, i \in \mathcal{V}_s\}$. For learning information from neighbor nodes, we adapt the GCNII [23] to achieve graph learning as following (9), where $\mathcal{N}_i$ is the neighbor timing wire set of timing wire i:

$$x_i^N = \mathrm{GCNII}_{\theta^i}\big(\{x_{j_n}, j_n \in \mathcal{N}_i\}\big). \tag{9}$$

Finally, the state of our framework for timing wire i can be obtained by combining the path module output x_i^P and multihop module output x_i^N

$$s_i = x_i^P \| x_i^N. \tag{10}$$

C. RL Action

RL actions $A : \{a_i, i \in \mathcal{V}_s\}$ represent the chosen LACs applied to timing wires. These LACs, which include wire-by-wire and wire-by-constant replacements, are selected from our specially designed RL action spaces. The action spaces are constructed based on timing path-aware LAC candidates. Diverging from conventional methods that generate LAC candidates based on similarity scores and logic depth, our work identifies timing path-aware LAC candidates by *timing path-aware similarity score* and *worst-arrive time*. These two metrics fully consider the relationships between LACs and timing paths, ensuring a more nuanced and effective circuit approximation process.

Timing Path-Aware Similarity Score: The signal value of one wire under one input vector can only be propagated to some target bits. The error distance is influenced when inaccuracies occur on high bits, particularly the MSBs. To optimize circuit timing, the LAC candidates are strategically placed on critical timing paths. Therefore, ensuring that LAC candidates on critical paths maintain values consistent with the target wire

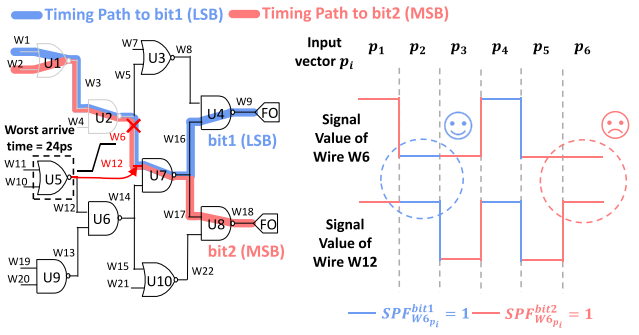

Fig. 7. Example of timing path-aware similarity score and worst-arrive time. Wire W6 is the target wire and wire W12 is the timing path-aware LAC candidate. Under different input vectors p_i, the signal values of wire W6 can be propagated to the least significant bit *bit*1 (LSB) when $\mathrm{SPF}^{bit1}_{W6_{p_i}} = 1$; and can be propagated to the MSB *bit*2 when $\mathrm{SPF}^{bit2}_{W6_{p_i}} = 1$. While comparing the signal values of wire W6 and W12, it is more important for them to be the same when $\mathrm{SPF}^{bit2}_{W6_{p_i}} = 1$ under error distance constraints.

is crucial. Such alignment plays a key role in meeting error distance constraints, especially when the signal is propagated to high-value bits.

As shown in Fig. 7, the signal value of wire W6 can be propagated to the MSB *bit*2 through the red path under input vectors $\{p_1, p_3, p_5, p_6\}$ and the low-significant bit *bit*1 through the blue path under input vectors $\{p_2, p_4\}$. To guarantee low-error distance, it is more important for the wire W12 signal to hold the same value with wire W6 under input vectors $\{p_1, p_3, p_5, p_6\}$ rather than under all input vectors. However, the weights of different bits are considered equally while computing similarity scores in previous works (1). In our work, different bits are weighted with their weight value [details described in (11)]. In addition, the signal propagation flag (SPF) is introduced to determine whether one wire signal value changes when one input vector is propagated to one specific bit. As shown in Fig. 7, SPF equals 1 when the signal change can be propagated to one specific bit. Otherwise, it equals 0.

The progress can be cast as the Boolean satisfiability problem for computing SPF values. The method in [3] based on Boolean difference can help obtain SPF values efficiently. The timing path-aware similarity score T represents the similarity score of one LAC weighted by bits and SPF values. It can be obtained through logic simulation where the higher value means the lower-error distance (NMED).

Worst-Arrive Time: Large-scale circuits have no clear relationship between smaller logic depth and smaller signal arrival time. This is because that even a few cells with large delay can cumulatively result in substantial signal arrival time at an LAC. To address this, our approach considers timing path information to identify timing path-aware LAC candidates. These candidates are pinpointed based on the worst-arrive time D, which represents the propagation delay from the inputs to the LAC candidate. Obtained via aging-and-variation-aware timing analysis, a smaller value indicates a greater potential for timing improvement.

Examples: Fig. 7 gives an example to compute timing path-aware similarity score and worst-arrival time. For wire W6, the timing-path aware similarity score of its candidate W12 ($T^{\mathrm{W6}}_{\mathrm{W12}}$) can computed as

$$T^{\mathrm{W6}}_{\mathrm{W12}} = \sum_{p_i}^{N_{p_i}} \sum_{\mathrm{bit}}^{N_{\mathrm{bit}}} \frac{(\mathrm{W6}_{p_i} == \mathrm{W12}_{p_i})? \times \mathrm{SPF}^{bit}_{\mathrm{W6}_{p_i}} \times 2^{bit}}{N_{p_i} \times 2^{MSB}} \quad (11)$$

where "$(\mathrm{W12}_{p_i} == \mathrm{W6}_{p_i})$?" equals to 1 if the signal value of W12 is the same with W6 signal value under input vector p_i. Otherwise, it equals to 0. $\mathrm{SPF}^{bit}_{\mathrm{W6}_v}$ is SPF of the specific bit *bit* for wire W6. 2^{bit} and 2^{MSB} are the weight values of the specific bit *bit* and MSB, respectively. N_{p_i} and N_{bit} are the number of input vectors and bits. MSB is the MSB value. The worst-arrival time of W15 equals 24ps, which is collected by aging-and-variation-aware STA.

The generation of the timing path-aware LAC candidate set C relies on *timing path-aware similarity scores* T and *worst-arrive time* D. In our approach, for each selected timing wire, we include the top-20 candidates based on the timing path-aware similarity scores in the candidate set C. The actions $A : \{a_i, i \in \mathcal{V}_s\}$ are expressed as

$$a_i = (T, D), i \in \mathcal{V}_s. \quad (12)$$

We construct the action spaces as continuous spaces based on C, offering two main advantages.

1) Compared to using discrete LAC candidates, our constructed action spaces can more effectively mine the relationship between LAC candidates and timing paths based on domain knowledge. The value of T impacts the error distance (NMED) and D influences the CPD.
2) The continuous nature of our action space allows for a broader range of possibilities in finding the optimal solution. However, the discrete action causes relative order information loss and overburdens the RL agent [24].

By translating the discrete action space into a continuous one, we ensure the RL agent's actions A are mapped effectively to the timing path-aware LAC candidate set C. After this mapping, we generate approximate circuits by assigning LACs selected from the LAC candidate set C.

D. RL Reward

RL reward R represents the CPD reduction under the error distance constraint. As mentioned in Section II-C, the timing and logic error evaluator can give us the accurate value of critical path delay CPD and error distance NMED fast under the aging effect and process variations. Then, the RL reward R is computed as

$$R = \begin{cases} \frac{(\mathrm{CPD}_{\mathrm{acc}} - \mathrm{CPD})}{\mathrm{CPD}_{\mathrm{acc}}} & \mathrm{NMED} \le \mathrm{NMED}_{\mathrm{con}} \\ \frac{(\mathrm{NMED}_{\mathrm{con}} - \mathrm{NMED})}{\mathrm{NMED}_{\mathrm{con}}} & \text{otherwise} \end{cases} \quad (13)$$

where $\mathrm{CPD}_{\mathrm{acc}}$ represents the CPD of the given accurate circuit. $\mathrm{NMED}_{\mathrm{con}}$ represents the NMED constraint, which can be changed with requirements. When the error distance of the approximate circuit is smaller than the constraint value $\mathrm{NMED}_{\mathrm{con}}$, we assign the reward based on the change in normalized CPD. Specifically, the CPD reduction means a positive reward. Conversely, if the error distance exceeds $\mathrm{NMED}_{\mathrm{con}}$, a negative reward is assigned as a penalty, proportionate to the error distance, to mitigate unacceptable accuracy loss.

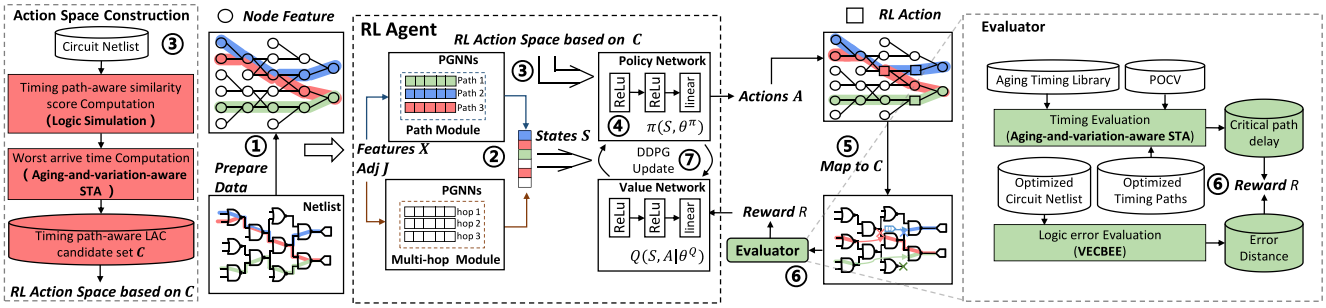

Fig. 8. Overview of RL-driven technology mapping approximation framework, including: 1) prepare data; 2) select and Embed RL states S; 3) generate timing path-aware LAC candidate set C and Construct RL action spaces based on C; 4) determine RL actions A; 5) map A to C; 6) compute RL rewards R; and 7) update the policy.

E. Overall Flow

In our RL agent, we employ DDPG [24], a variant of actor-critic algorithms, to determine the RL actions A based on RL states S and RL reward R. In actor-critic algorithms, there are two key components: 1) actor and 2) critic. Within deep RL, which integrates neural networks, the actor is the policy network $\pi(S \mid \theta^{\pi})$. This network learns a parameterized policy that maps all states S in one episode to actions A. The critic is the value network $Q(S, A \mid \theta^{Q})$, responsible for learning a value function that evaluates the reward R resulting from taking actions A on states S. Notably, our approach is end-to-end, meaning that the PGNNs, the policy network, and the value network are trained jointly, enhancing their generalization ability.

The details of the training process are illustrated in Algorithm 1. Graph learning is achieved first to collect circuit structure and timing path information (lines 5–7). During warm-up episodes, we randomly sample and store some transitions in the replay buffer F (lines 8–12), which contains old experiences from previous episodes. Then, the training target of our value network Q is to make precise reward predictions based on varying states, utilizing a loss function based on the cross-entropy of predicted rewards and ground truth simulations. The policy network π aims to generate the action, subject to the state that maximizes the Q-value that maximize the Q-value, with its loss function being the gradient of the reward relative to actions (lines 17–18). Upon completion of training, we achieve a finely tuned value network and an effective policy network, ready for design automation to optimize timing.

Once the RL agent is trained for technology mapping approximation, an overview of the proposed framework is shown in Fig. 8. In each iteration: 1) prepare data for the given accurate circuit graph $\mathbb{G} = (\mathcal{V}, \mathcal{E})$, including timing wire feature matrices X and adjacency matrix J; 2) select timing wires to generate $\mathcal{V}_s$ and embed feature vectors for selected timing wires to RL sates $S : \{s_i, i \in \mathcal{V}_s\}$ using PGNNs; 3) generate timing path-aware approximate candidate set C and construct continuous RL action space based on C; 4) determine the RL actions $A : \{a_i, i \in \mathcal{V}_s\}$ in the policy network; 5) map the actions A to timing path-aware approximate candidates C for generating approximate circuit; 6) compute the RL reward R of the generated approximate circuit after taking actions A on states S using timing and logic error evaluator; Here, timing analysis focuses on optimized paths; and 7) update the RL actions A based on RL reward R and states S.

Algorithm 1 RL Agent Training Methodology

Input: Circuit graph: $\mathbb{G} = (\mathcal{V}, \mathcal{E})$; Node feature matrix: $X : \{x_k, \forall k \in V\}$; Adjacency matrix: J; Number of sampled data batch: N_s; Exponential moving parameter: B; Maximum search episodes: M; Warm-up episodes: M_w; Replay buffer: F.
Output: PGNN parameters: θ^g, θ^a and θ^i; Policy Network parameters: θ^{π}; Value Network parameters: θ^{Q}.

1: $\{\mathcal{V}_s\}$ ← Select independent timing wires on potential critical paths in $\mathbb{G}$ (shown in Section III-B);
2: $\{C\}$ ← Generate timing path-aware LAC candidate set (shown in Section III-C);
3: Construct continuous action spaces based on $\{C\}$;
4: **for** *episode* ← 1 to M **do**
5: **for** $i \in \mathcal{V}_s$ **do**
6: s_i ← Embed x_i to generate state s_i using PGNNs;
7: **end for**
8: Receive observation states S;
9: Sample a set of actions A randomly;
10: Map actions A to C then obtain the approximate circuit;
11: Compute reward R of the circuit using Equation (13);
12: Store transition (S, A, R) in F;
13: **if** *episode* $> M_w$ **then**
14: Sample a batch of $(\widehat{S}, \widehat{A}, \widehat{R})$ from F;
15: Update θ^{Q} by minimizing the loss L:
16: $\frac{1}{N_s}\sum_{k=1}^{N_s}\left(\widehat{R}^k - B - Q\left(\widehat{S}^k, \widehat{A}^k \mid \theta^{Q}\right)\right)^2$
17: Update θ^{π} based on policy gradient $\nabla_{\theta^{\pi}} J$:
18: $\frac{1}{N_s}\sum_{k=1}^{N_s}\nabla_{\theta^{\pi}} Q\left(\widehat{S}^k, \left(\pi\left(\widehat{S}^k\right) \mid \theta^{\pi}\right) \mid \theta^{Q}\right)$
19: **end if**
20: **end for**

IV. Experimental Results

The timing-driven technology mapping approximation framework is implemented in Python with the Pytorch library and trained on a Linux machine with 32 cores and four NVIDIA Tesla V100 GPUs. The maximum search episodes M and warm-up episodes M_w are set to be 400 and 100. The train-

TABLE II
ARITHMETIC BENCHMARK STATISTICS. #P.PATHS REPRESENTS THE NUMBER OF POTENTIAL CRITICAL PATHS SELECTED IN THE FIRST OPTIMIZATION ITERATION. THE NUMBER OF TIMING PATH-AWARE LAC CANDIDATES FOR EACH SELECTED WIRE ON POTENTIAL CRITICAL PATHS EQUALS 20

Circuit	#gate	#wire	#input	#output	#p.paths	Description
Adder	1252	1123	256	129	12604	128-bit adder
Bar	2933	2805	135	128	12800	128-bit shifter
Div	30047	29919	128	128	12800	128-bit divisor
Log2	29279	29310	32	32	3200	32-bit log2 unit
Max	2296	2166	512	120	12908	128-bit 4-1 max unit
Multiplier	24536	24562	128	128	12688	128-bit multiplier
Sin	6061	6039	24	25	2500	24-bit sine unit
Sqrt	13348	13284	128	64	6158	128-bit square root unit
Square	14790	14756	64	128	12330	64-bit square unit
Total	233100	239395	1663	1020	100788	Arithmetic circuits

TABLE III
RANDOM/CONTROL BENCHMARK STATISTICS. #P.PATHS REPRESENTS THE NUMBER OF POTENTIAL CRITICAL PATHS SELECTED IN THE FIRST OPTIMIZATION ITERATION

Circuit	#gate	#wire	#input	#output	#p.paths	Description
Arbiter	6603	6859	256	129	12900	Round-robin arbiter
ALU	141	148	7	26	450	Alu control unit
Cavlc	671	681	10	11	902	Coding Cavlc
Router	168	228	60	30	300	Lookahead router
Decode	360	368	8	256	4096	Decoder
I2C	1049	1196	147	142	5672	I2C controller
Conv.	206	217	11	7	492	Value converter
Memory	19665	20869	1204	1231	67850	Memory controller
Voter	6466	7499	1001	1	100	Voter unit
Total	35163	38017	2822	1830	92660	Random/Control circuits

ing process of our RL agent takes about 12.8 h using parallel training method A3C [25] on 4 GPUs. We train TSMC 28-nm and TSMC 16-nm RL frameworks using 28 and 16-nm standard arithmetic circuits, including different bit-width adders, multipliers, and divisors. Reward shaping helps meet NMED, ER, and MRED constraints in different frameworks.

To demonstrate the effectiveness and generalization ability, we apply our trained framework to nine EPFL arithmetic and nine EPFL random/control circuits [26] without retraining. The benchmark statistics are shown in Tables II and III. All accurate circuits are synthesized with TSMC 28 and 16-nm technology using Synopsys Design Compiler [27]. Note that all generated approximate circuits can obtain different timing and area improvements. In our work, we focus on timing improvements. For fair comparisons, the generated approximate circuits are post-optimized in the Design Compiler under area constraints without adjusting any circuit structure to compare the timing improvements. The used area constraints are shown in Table V.

The timing path-aware similarity scores are generated by logic simulation based on Mentor Modelsim [28]. The CPDs and worst-arrival time are obtained through aging-and-variation-aware STA using Synopsys PrimeTime [29] with aging-aware timing library [17] and variation-aware timing library [19]. During timing evaluation, the command timing_pocvm_report_sigma 3 is applied to ensure that critical paths are selected after considering enough timing guardbands. Due to no specific application, the signal probabilities of all gates are computed based on logic simulation when the signal probabilities of all inputs are set to 0.5. Fig. 9 shows distributions of signal probability in the divisor and the log2 unit. During logic error evaluation, the NMEDs, ERs, and

TABLE IV
ERROR DISTANCE RESULTS CAUSED BY TIMING ERRORS UNDER DIFFERENT TIMING GUARDBANDS

Circuit	$NMED$ (%) under different timing guardbands			
	$\mu(\mathcal{D})$	$\mu(\mathcal{D})+\sigma(\mathcal{D})$	$\mu(\mathcal{D})+2\sigma(\mathcal{D})$	$\mu(\mathcal{D})+3\sigma(\mathcal{D})$
Adder	0.512	0.032	5.291×10^{-4}	1.349×10^{-7}
Bar	0.785	0.187	1.873×10^{-2}	9.845×10^{-6}
Div	9.537	1.243	8.942×10^{-3}	5.142×10^{-5}
Log2	12.021	6.354	8.163×10^{-3}	4.475×10^{-6}
Max	8.825	2.649	3.592×10^{-2}	1.832×10^{-5}
Multiplier	6.774	0.953	5.564×10^{-3}	2.443×10^{-5}
Sin	1.092	0.239	2.342×10^{-3}	8.653×10^{-6}
Sqrt	2.572	0.978	5.213×10^{-4}	3.441×10^{-7}
Square	6.265	1.998	2.878×10^{-3}	1.496×10^{-6}

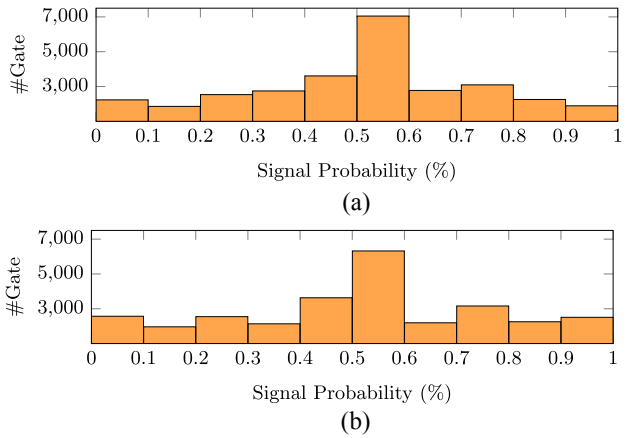

Fig. 9. Signal probability distribution of gates in the divisor and the Log2 unit. (a) 128-bit divisor. (b) 32-bit log2 unit.

MREDs are calculated through VECBEE [3]. They can be obtained based on Monte Carlo simulation to input vectors. In terms of the number of input vectors N_{p_i}, we follow the setting in [3] where $N_{p_i} = 100\ 000$, which can give a high-accuracy error estimation.

For the 28-nm framework, we compare the timing optimization efficiency of our framework, including optimization results and runtime, with 1) the genetic algorithm-based method [8], [9]; 2) VECBEE-SASIMI [3] using the greedy method; 3) a simple RL agent using LAC candidates; 4) a modified RL agent using timing path-aware LAC candidates without GNNs; and 5) an RL agent equipped using timing path-aware LAC candidates equipped with GCNII [23].

For the 16-nm framework, we compare the timing optimization efficiency of our framework with 1) the genetic algorithm-based method [8], [9]; 2) VECBEE-SASIMI [3]; 3) PowerX [6] using deep learning to predict ER; 4) SEALS using [5] sensitivity to achieve fast and accurate error estimation; and 5) HEDALS [4] on critical timing paths. For head-to-head comparisons, the objectives of all works are optimizing CPD under error constraints, including NMEDs, ERs, and MREDs.

A. Timing Errors Under Different Timing guardbands

Timing guardbands are essential to cover aging and variation-induced timing degradations. Table IV gives the NMED results of EPFL arithmetic circuits under various timing guardbands and without guardbands considering. A 5-year

TABLE V
CPD UNDER FRESH CONDITION CPD-0, CPD UNDER 5-YEAR-AGED CONDITION CPD-5 OF APPROXIMATE CIRCUITS GENERATED BY OUR WORK AND OTHERS UNDER THE SAME ERROR DISTANCE CONSTRAINT $\text{NMED}_{\text{con}} = 0.196\%$. THE UNITS OF CPD-0 AND CPD-5 ARE PS. AREA CON. REPRESENTS THE AREA CONSTRAINTS WE USED TO RESYNTHESIZE APPROXIMATE CIRCUITS WITHOUT ADJUSTING CIRCUIT STRUCTURES AND THE UNIT OF IT IS μm^2. THE NMEDS OF ALL APPROXIMATE CIRCUITS ARE A LITTLE SMALLER THAN OR EQUAL TO 0.196%

Circuit	Area con.	Accurate		Genetic Algo. [8], [9]		V-SASIMI [3]		SimpleRL		ModifyRL		GCNII-RL		Ours	
		$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$
Adder	495.00	1394.7	1692.4	1156.2	1386.1	1108.8	1309.9	1163.2	1391.2	1097.6	1271.0	1083.7	1260.8	**965.1**	**1137.3**
Bar	1800.00	560.0	639.8	527.0	596.9	517.4	585.4	535.4	600.8	497.3	560.5	490.0	554.1	**438.5**	**493.9**
Div	13000.00	52499.8	57361.9	51817.3	55239.5	51187.3	54952.7	52027.3	55927.9	48877.3	53231.8	48509.8	52715.6	**43312.3**	**47208.8**
Log2	17000.00	5824.0	6352.1	5789.1	6174.2	5748.3	6009.1	5806.5	6244.1	5486.2	5742.3	5387.2	5666.1	**4985.3**	**5329.4**
Max	954.00	2799.8	3192.9	2545.0	2854.5	2517.0	2828.9	2556.2	2828.9	2416.2	2720.4	2360.2	2666.1	**2175.4**	**2397.9**
Mul	15000.00	3070.0	3586.9	2520.5	2880.3	2314.8	2675.8	2560.4	2926.9	2250.3	2586.2	2238.0	2561.0	**1970.9**	**2256.2**
Sin	4300.00	3050.0	3629.4	2943.3	3477.0	2876.2	3382.6	2952.4	3495.1	2787.7	3219.3	2751.1	3172.1	**2324.1**	**2878.1**
Sqrt	6200.00	118998.6	135322.4	90676.9	100815.2	86750.0	95943.6	94960.9	106769.4	83775.0	92966.5	82466.0	91883.9	**72946.1**	**82140.7**
Square	7700.00	1540.0	1792.1	1295.1	1499.0	1242.8	1408.6	1312.1	1516.1	1193.5	1365.6	1178.1	1356.6	**1085.7**	**1241.9**
Average	7383.22	21081.9	23729.9	17696.7	19435.9	17140.3	18788.5	18208.3	20188.9	16486.8	18184.8	16273.8	17981.8	**14467.1**	**16120.5**

aging condition is applied. $\mathcal{D}$ is the delay distribution of the most critical path after considering aging and variations. And $\mu(\mathcal{D})$ and $\sigma(\mathcal{D})$ represent mean and standard deviation values of the delay distribution $\mathcal{D}$, respectively. According to the results, $\mu(\mathcal{D}) + 3\sigma(\mathcal{D})$ can avoid the majority of timing errors in our experiments.

B. Optimization Results of the 28-nm Framework

Table V demonstrates the timing results under the error distance constraint NMED_{con} set to $(20/2^{10} - 1) \approx 0.196\%$. We observe that the approximate circuits generated by our framework can outperform other state-of-the-art works. The critical path delay of the fresh (CPD-0) and the 5-year-aged condition (CPD-5) are examined for each circuit. All test algorithms are terminated when the 5-year-aged critical path delay (CPD-5) no longer decreases or the error distance (NMED) exceeds the constraint across 10 consecutive iterations, using CPD-5 in our RL reward *R*. Overall, our framework achieves an average 25.41% CPD reduction under the fresh condition and an average 26.92% critical path reduction under the 5-year-aged condition. It means the aging-and-variation-induced timing guardbands of accurate circuits can be eliminated effectively through our work. After analyzing the results, we summarize our findings below.

1) We use the trained RL agent to achieve timing optimization on unseen circuits. The results suggest that our work can generalize across various designs with different functions and scales.
2) We achieve significant reductions of CPD under error distance constraints. It means approximate circuits generated by our work are fast and accurate.
3) Compared with the genetic and greedy method, our RL-based work can achieve much better-timing optimization performance after constructing reasonable action space and considering timing information.
4) Comparing the modified RL agent with the simple RL agent, the modified work can achieve larger timing optimization. Thus, the effectiveness of timing path-aware LAC candidates can be proved.
5) Compared with the GCNII-RL agent, our work achieves better-optimization performance benefiting from modeling timing paths accurately in our RL state representations.

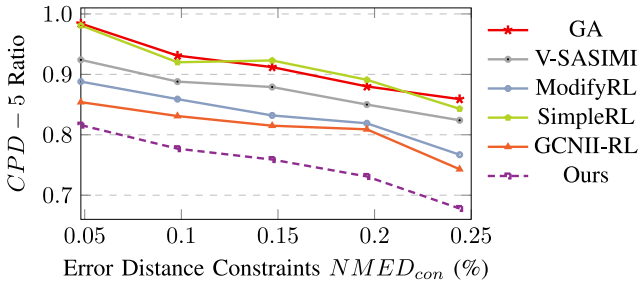

Fig. 10. Average critical path delay CPD-5 ratios of the approximate circuits obtained by our work and others over the accurate circuits under different error distance constraints NMED_{con}.

We compare the performance of our framework with others under different error distance constraints NMED_{con}, including $(5/2^{10} - 1) \approx 0.048\%$, $(10/2^{10} - 1) \approx 0.098\%$, $(15/2^{10} - 1) \approx 0.147\%$, $(20/2^{10} - 1) \approx 0.196\%$, and $(25/2^{10} - 1) \approx 0.244\%$. In Fig. 10, we illustrate how the average CPD ratio varies with these error distance constraints. The results indicate that our generated approximate circuits consistently achieve greater reductions in CPD. This signifies that our framework can attain higher levels of timing optimization while meeting diverse functional accuracy requirements.

To illustrate the effectiveness of the approximated circuits generated by our work across the entire design space, we compare the timing improvements under a range of area constraints. Fig. 11 shows a comparison of CPD versus area constraints for approximate circuits developed by different methods when $\text{NMED}_{\text{con}} = (20/2^{10} - 1) \approx 0.196\%$. The approximate circuits generated by our work outperform other works across all area constraints, showcasing the breadth and effectiveness of our optimizations. This achievement highlights our framework's ability to fulfill diverse design requirements with superior timing optimization performance.

The focus of our work is on achieving timing optimization within error constraints. Interestingly, this process also tends to result in power reduction due to the nature of the approximations applied. PrimeTime measures the power consumption after gate-level simulation. The results of different approximated circuits are shown in Table VI. It is noted that aging effects typically increase transistor threshold voltage, leading to reductions in both leakage and dynamic power consumption [8]. Consequently, the power consumption of aged circuits is generally lower than that of their fresh counterparts.

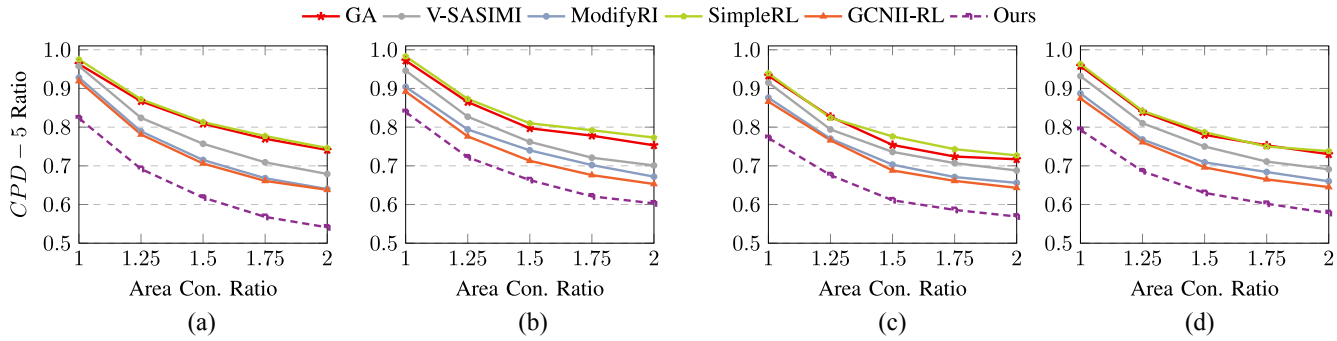

Fig. 11. Critical path delay CPD-5 ratios of the approximate circuits generated by different works over different area constraints for four different designs. Area Con. Ratio represents the ratio of new area constraints over original area constraints shown in Table V. (a) Divisor. (b) Log2 unit. (c) Shifter. (d) Sine unit.

TABLE VI
POWER CONSUMPTION UNDER FRESH CONDITION *POW*-0, POWER CONSUMPTION UNDER 5-YEAR-AGED CONDITION *POW*-5 OF APPROXIMATE CIRCUITS GENERATED BY OUR WORK AND OTHERS UNDER THE SAME ERROR DISTANCE CONSTRAINT $\text{NMED}_{\text{con}} = 0.196\%$. THE UNITS OF *POW*-0 AND *POW*-5 ARE *mW*

Circuit	Area con.	Accurate		Genetic Algo. [8], [9]		V-SASIMI [3]		SimpleRL		ModifyRL		GCNII-RL		Ours	
		$POW-0$	$POW-5$	$POW-0$	$POW-5$	$POW-0$	$POW-5$	$POW-0$	$POW-5$	$POW-0$	$POW-5$	$POW-0$	$POW-5$	$POW-0$	$POW-5$
Adder	495.00	4.6928	4.5876	4.3174	4.3078	3.9983	3.9499	4.5145	4.4683	4.4629	4.4133	4.2094	4.1793	4.0077	3.9407
Bar	1800.00	2.4982	2.3841	2.2484	2.1958	2.0061	1.9573	2.3433	2.2840	2.3183	2.2458	2.1584	2.0003	2.0160	1.9597
Div	13000.00	0.0558	0.0557	0.0528	0.0530	0.0470	0.0474	0.0553	0.0555	0.0550	0.0550	0.0503	0.0509	0.0468	0.0473
Log2	17000.00	1.9617	1.8969	1.8440	1.8002	1.5929	1.5611	1.9146	1.8722	1.8715	1.8419	1.7243	1.6731	1.6106	1.5782
Max	954.00	0.5742	0.5429	0.5386	0.5163	0.4645	0.4408	0.5432	0.5331	0.5352	0.5217	0.4904	0.4674	0.4663	0.4430
Mul	15000.00	5.0639	4.8453	4.5676	4.5449	4.0764	3.9295	4.8158	4.7484	4.6740	4.5740	4.2739	4.1573	4.0866	3.9586
Sin	4300.00	0.8754	0.8497	0.8115	0.8055	0.7205	0.7061	0.8500	0.8361	0.8185	0.8089	0.7651	0.7503	0.7161	0.7027
Sqrt	6200.00	0.0578	0.0576	0.0546	0.0548	0.0472	0.0475	0.0564	0.0570	0.0550	0.0557	0.0501	0.0504	0.0476	0.0479
Square	7700.00	5.0889	4.8567	4.5647	4.4876	4.2340	4.0602	4.7887	4.6576	4.7378	4.5799	4.6411	4.4876	4.2085	4.0408
Average	7383.22	2.3187	2.2307	2.1111	2.0851	1.9096	1.8556	2.2091	2.1680	2.1698	2.1218	2.0403	1.9796	1.9118	1.8577

TABLE VII
CPD UNDER FRESH CONDITION CPD-0, CPD UNDER 5-YEAR-AGED CONDITION CPD-5 OF APPROXIMATE CIRCUITS GENERATED BY OUR WORK AND OTHERS USING *TSMC 16-nm Technologies* UNDER THE SAME ERROR DISTANCE CONSTRAINT $\text{NMED}_{\text{con}} = 0.196\%$. THE NMEDS OF ALL APPROXIMATE CIRCUITS ARE A LITTLE SMALLER THAN OR EQUAL TO 0.196%

Circuit	Area con.	Accurate		Genetic Algo. [8], [9]		V-SASIMI [3]		PowerX [6]		SEALS [5]		HEDALS [4]		Ours	
		$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$
Adder	560.00	391.4	513.9	329.6	431.2	322.1	418.3	335.0	436.3	319.0	412.1	290.4	369.5	**265.2**	**336.3**
Bar	725.00	182.8	272.3	176.0	259.2	173.3	253.8	178.8	262.5	168.5	246.4	153.4	220.3	**138.3**	**202.3**
Div	19400.00	4356.1	5686.5	4338.7	5578.5	4277.7	5504.5	4334.3	5584.1	4194.9	5436.3	3750.6	4771.0	**3554.3**	**4621.1**
Log2	7750.00	1065.8	1686.7	1063.7	1664.8	1056.2	1604.1	1064.7	1671.5	1048.7	1590.6	949.6	1438.8	**896.7**	**1403.3**
Max	820.00	385.9	531.9	359.3	486.7	351.9	481.4	357.3	483.5	348.9	475.0	325.7	429.2	**297.1**	**387.6**
Mul	6500.00	553.7	712.3	471.8	588.4	449.6	570.6	493.9	609.7	441.9	557.0	407.5	519.3	**348.2**	**434.1**
Sin	1700.00	582.6	769.1	570.4	739.1	554.1	722.2	560.5	721.4	547.1	716.8	495.8	629.1	**447.2**	**596.2**
Sqrt	11350.00	6038.8	8197.3	4988.0	6631.6	4843.1	6418.5	5018.2	6648.0	4764.6	6377.5	4257.4	5893.9	**3773.6**	**5007.2**
Square	3950.00	249.9	427.8	222.2	373.5	210.9	355.5	223.2	376.0	206.9	350.4	193.2	314.4	**172.5**	**287.9**
Average	5861.67	1534.1	2088.6	1391.1	1861.4	1359.9	1814.3	1396.2	1865.9	1337.8	1795.8	1202.6	1620.6	**1099.2**	**1475.1**

C. Optimization Results of the 16-nm Framework

The 16-nm frameworks for meeting different kinds of error constraints are trained based on TSMC 16-nm standard arithmetic circuits, including various bit-width adders, multipliers, and divisors. Table VII demonstrates the performance of approximate circuits generated by the 16-nm framework under the NMED constraint. The constraint NMED_{con} is set to $(20/2^{10}-1) \approx 0.196\%$. Our evaluation includes analyzing the critical path delay for both fresh (CPD-0) and the 5-year-aged condition (CPD-5) of each circuit. Notably, 16-nm technology exhibits greater sensitivity to process variations and aging effects than 28-nm technology, making it challenging for other methods to optimize timing effectively. However, our framework successfully overcomes this, achieving an average reduction of 27.59% in CPD-0 for fresh conditions and 28.92% in CPD-5 for aged conditions.

For other kinds of error constraints, the 16-nm framework is retrained by reward shaping. Table VIII demonstrates the timing results under ER constraint ER_{con} setting to 2% for random/control circuits. And Table IX demonstrates the timing results under error distance constraint $MRED_{\text{con}}$ setting to $(40/2^{10}-1) \approx 0.392\%$ for arithmetic circuits. Under error rate ER constraints, the approximate circuits generated by our framework achieve an average 21.93% CPD reduction under the fresh condition and an average 24.55% reduction under the 5-year-aged condition. Under relative mean error distance *MRED* constraints, the approximate circuits generated by our framework achieve an average 22.09% CPD reduction under the fresh condition and an average 30.65% reduction under the 5-year-aged condition.

Fig. 12(a) plots how the average CPD ratio changes with error distance constraints NMED_{con} under the TSMC 16-nm technology. The frameworks PowerX [6], VECBEE-SASIMI [3], SEALS [5], although expedient in error prediction, face limitations in their accuracy, impacting overall efficacy. HEDALS [4] optimizes timing performance through approximate computing on critical paths but tends to yield local optima due to its greedy algorithm and independent

TABLE VIII
CPD UNDER FRESH CONDITION CPD-0, CPD UNDER 5-YEAR-AGED CONDITION CPD-5 OF APPROXIMATE CIRCUITS GENERATED BY OUR WORK AND OTHERS UNDER THE SAME ER CONSTRAINT $ER_{\text{con}} = 2\%$. THE *ERs* OF ALL APPROXIMATE CIRCUITS ARE SLIGHTLY SMALLER THAN OR EQUAL TO 2%

Circuit	Area con.	Accurate		Genetic Algo. [8], [9]		V-SASIMI [3]		PowerX [6]		SEALS [5]		HEDALS [4]		Ours	
		$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$
Arbiter	2200.00	77.7	91.3	72.6	83.2	70.4	81.0	75.0	87.3	68.6	78.8	62.5	71.4	**56.7**	**62.5**
Cavlc	165.00	53.8	78.3	51.1	72.5	49.6	71.5	52.4	75.1	48.6	69.8	45.2	63.7	**40.5**	**59.7**
ALU	65.00	36.3	52.4	32.9	46.2	31.8	44.7	33.5	47.8	31.1	43.6	28.4	40.8	**25.7**	**37.2**
Router	55.00	52.0	73.7	45.7	62.8	44.4	61.2	46.9	65.8	43.8	60.5	40.2	56.2	**36.7**	**49.1**
Decode	115.00	44.1	62.8	35.9	50.3	34.3	47.5	37.3	52.3	33.2	45.5	31.4	44.0	**28.7**	**40.5**
I2C	360.00	49.4	65.4	43.6	56.3	41.3	53.1	44.1	57.1	40.1	52.4	36.7	47.9	**33.1**	**42.2**
Cov.	80.00	46.6	61.5	38.3	48.8	35.2	45.1	40.2	52.9	34.3	43.8	32.8	41.5	**30.6**	**39.7**
Memory	4800.00	165.3	238.1	163.8	233.3	162.9	229.5	164.3	235.3	152.4	212.9	141.5	197.9	**136.1**	**187.7**
Voter	2250.00	295.3	324.6	288.9	310.2	275.2	298.9	290.4	314.5	257.7	276.2	244.1	260.3	**233.1**	**244.8**
Average	1121.11	91.2	116.5	85.9	107.1	82.8	103.6	87.1	109.8	78.9	98.2	73.6	91.5	**69.0**	**84.8**

TABLE IX
CPD UNDER FRESH CONDITION CPD-0, CPD UNDER 5-YEAR-AGED CONDITION CPD-5 OF APPROXIMATE CIRCUITS GENERATED BY OUR WORK AND OTHERS UNDER THE SAME ERROR DISTANCE CONSTRAINT $MRED_{\text{con}} = 0.392\%$. THE *MREDs* OF ALL APPROXIMATE CIRCUITS ARE A LITTLE SMALLER THAN OR EQUAL TO 0.392%

Circuit	Area con.	Accurate		Genetic Algo. [8], [9]		V-SASIMI [3]		PowerX [6]		SEALS [5]		HEDALS [4]		Ours	
		$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$	$CPD-0$	$CPD-5$
Adder	560.00	391.4	513.9	337.8	441.4	324.5	417.3	349.5	453.3	318.2	413.2	288.1	365.9	**272.9**	**347.3**
Bar	725.00	182.8	272.3	180.6	266.6	171.5	251.1	182.3	267.9	165.3	243.2	154.5	226.3	**142.1**	**213.3**
Div	19400.00	4356.1	5686.5	4325.6	5584.1	4260.3	5459.0	4338.7	5623.9	4147.0	5328.3	3794.2	4930.2	**3711.9**	**4772.5**
Log2	7750.00	1065.8	1686.7	1064.7	1674.9	1055.1	1605.7	1065.8	1681.6	1036.0	1626.0	972.0	1516.3	**932.7**	**1456.3**
Max	820.00	385.9	531.9	358.1	487.8	348.1	473.9	365.1	528.2	336.1	454.2	320.7	435.6	**303.9**	**417.2**
Mul	6500.00	553.7	712.3	491.1	624.0	466.8	584.8	510.0	647.5	460.1	578.4	419.7	520.7	**386.3**	**479.1**
Sin	1700.00	582.6	769.1	574.4	747.6	554.1	724.5	579.1	758.3	538.3	689.1	483.0	623.7	**445.9**	**583.4**
Sqrt	11350.00	6038.8	8197.3	5126.9	6754.6	4674.0	6107.0	5350.4	7139.8	4583.4	6041.4	4378.1	5893.9	**4315.3**	**5742.1**
Square	3950.00	249.9	427.8	224.2	374.8	212.9	358.1	233.4	393.1	201.4	334.1	197.2	328.6	**182.3**	**308.6**
Average	5861.67	1534.1	2088.6	1409.3	1884.0	1340.8	1775.7	1441.6	1943.8	1309.5	1745.3	1223.0	1649.0	**1188.2**	**1591.1**

TABLE X
OPTIMIZATION RUNTIME (MIN.) COMPARISON

Circuit	Genetic	V-SASIMI	SimpleRL	ModifyRL	GCNII-RL	Ours
Adder	20.51	28.49	**7.29**	13.09	12.68	8.91
Bar	32.14	39.79	18.34	20.98	24.57	**17.30**
Div	859.16	1182.91	312.51	502.16	232.92	**193.86**
Log2	681.2	702.24	209.89	189.24	165.83	**135.05**
Max	27.21	34.95	21.39	24.29	18.41	**15.29**
Mul	213.42	224.25	78.92	149.57	61.29	**46.72**
Sin	48.20	57.61	34.19	48.91	42.13	**38.40**
Sqrt	543.23	741.98	279.82	306.86	189.71	**151.43**
Square	38.09	44.59	**21.59**	30.58	32.34	26.23
Average	273.68	339.65	109.33	142.85	86.65	**70.36**

path-by-path consideration. Our RL-based framework, by contrast, generates approximate circuits with greater reductions in CPD by incorporating a more comprehensive analysis of timing path information. Fig. 12(b) and (c) plot how the average CPD ratio changes with error distance constraints ER_{con} and $MRED_{\text{con}}$, respectively. In summary, our work can achieve more CPD reduction when compared with other works under different error constraints.

D. Optimization Runtime

Table X demonstrates the runtime for timing optimization across different methods under error distance constraints, showing that our work optimizes the smallest design (Adder) in 8.91 min and the largest one (Div) in 193.86 min. Our method is slightly more time-consuming than others in one iteration due to accurate aging-and-variation-aware timing analysis on a limited number of optimized paths using PrimeTime [29]. As demonstrated in Fig. 13, the bulk of runtime in one iteration about 70% is consumed by logic error and timing evaluation for both HEDALS [4] and our work. Thus, the overall runtime is predominantly influenced by the convergence speed of the optimization algorithms. Our framework benefits from an accelerated convergence speed, resulting in faster optimization and more enhanced scalability, which is particularly advantageous for large-scale circuits. This acceleration is achieved through a well-designed action space, an optimal optimization policy, and applying multiple LACs in each iteration.

E. Statistics on LACs

Fig. 14 displays the average percentage of different types of LACs, including wire-by-constant and wire-by-wire replacements, across various circuits under NMED, ER, and *MRED* constraints. It reveals that wire-by-wire replacements are more prevalent, constituting 67.6%, 59.8%, and 70.5% under NMED, ER, and *MRED* constraints, respectively. The lower prevalence of wire-by-wire replacements under ER constraints is attributed to the relative ease of maintaining low ERs compared to error distances in design automation. Moreover, since wire-by-constant replacements tend to offer larger timing optimization benefits, their usage increases under ER constraints.

F. Why Our RL-Based Work Is Effective?

The experimental results show that the timing performance of approximated circuits generated by our work under different error constraints outperforms all other methods. As shown in Table XI, we compare our work with other methods. In summary, the improved efficacy of our work is achieved by the following.

1) *More Collected Information:* First, the circuit structure information is important to avoid an accuracy loss that is too high after approximate computing. The structure

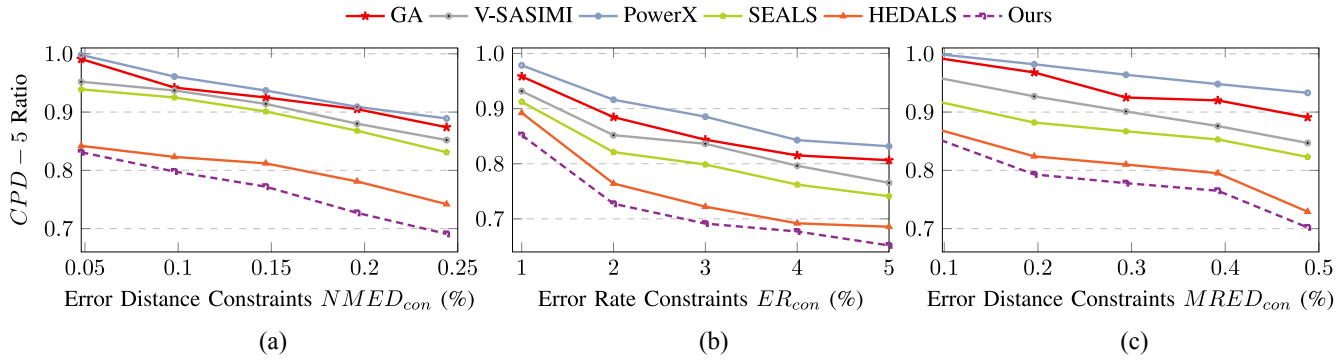

Fig. 12. Average critical path delay CPD-5 ratios of the approximate circuits obtained by our work and others over the accurate circuits using *TSMC 16-nm technology* under different error constraints (a) NMED_{con}; (b) ER_{con}; (c) $MRED_{\text{con}}$.

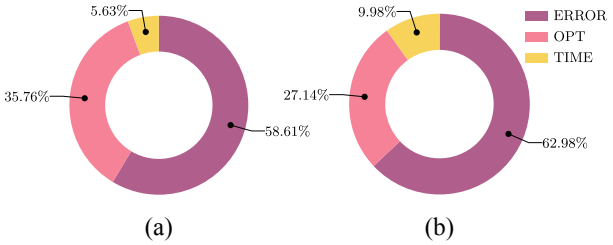

Fig. 13. Runtime breakdown in one iteration of (a) HEDALS [4] and (b) our work.

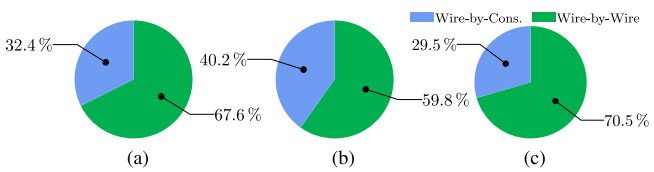

Fig. 14. Average percentage of different LACs under different error constraints: (a) NMED_{con}; (b) ER_{con}; and (c) $MRED_{\text{con}}$.

information about relationships between one gate and output is defined as sensitivity in SEALS [5]. The simple deep learning method [6] cannot handle circuit structure while it is collected via graph learning efficiently in our work. In addition, timing information on timing paths is critical for optimization. Among previous works, HEDALS [4] is the only one considering timing path information. However, it cannot consider timing information on multiple timing paths jointly where one LAC on one path can achieve timing optimization, which might cause timing degradation on other paths due to load increase. Finally, the relationship between timing paths and LACs should be mined and used. However, it has been ignored in previous works.

2) *Better-Optimization Strategy:* The solution space of approximate logic optimization and technology mapping approximation is always very large. Dynamic optimization can solve it more efficiently than static methods. Traditional algorithms used in other works, including greedy and genetic algorithms, are constructed based on static optimization. RL is based on the Markov decision process, which solves the optimization problem in a dynamic way [30]. In addition, the trained RL framework outputs an optimal policy network

TABLE XI
WORK COMPARISON FROM DIFFERENT VIEWS

Work	Circuit Sturcture	Timing Paths	Path-aware LACs	Multiple LACs	Optimization Method
PowerX [6]	✘	✘	✘	✘	Greedy
KIT [8], [9]	✘	✘	✘	✘	Genetic
V-SASIMI [3]	✘	✘	✘	✘	Greedy
SEALS [5]	✔	✘	✘	✘	Greedy
HEDALS [4]	✘	✔	✘	✘	Greedy
Ours	✔	✔	✔	✔	RL

rather than just optimal actions generated via traditional optimization methods. It helps us to obtain stable performance. More importantly, many RL frameworks that focus on solving similar problems can share knowledge to improve their efficiency. In our work, similar problems are achieving timing optimization under different error constraints and different technology nodes.

V. CONCLUSION AND FUTURE WORKS

This work proposes and implements a timing-driven technology mapping approximation framework based on RL to optimize aging-and-variation-aware timing under error distance constraints. The high-efficiency and powerful generalization ability of our RL framework come from two key strategies: 1) accurately modeling timing paths while embedding RL states using PGNNs and 2) considering the relationship between LACs and timing paths when constructing RL action spaces. Experimental results with open-source designs demonstrate the superior efficiency of our framework over existing approaches, both in results and runtime. While this work concentrates on technology mapping approximation, future endeavors aim to integrate approximate logic optimization for enhanced timing optimization.

REFERENCES

[1] R. Huang et al., "Variability-and reliability-aware design for 16/14 nm and beyond technology," in *Proc. IEEE Int. Electron Devices Meet. (IEDM)*, 2017, pp. 12.4.1–12.4.4.

[2] Z. Zhang, Z. Guo, Y. Lin, R. Wang, and R. Huang, "AVATAR: An aging- and variation-aware dynamic timing analyzer for application-based DVAFS," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 841–846.

[3] S. Su et al., "VECBEE: A versatile efficiency–accuracy configurable batch error estimation method for greedy approximate logic synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, vol. 41, no. 11, pp. 5085–5099, Nov. 2022.
[4] C. Meng, Z. Zhou, Y. Yao, S. Huang, Y. Chen, and W. Qian, "HEDALS: Highly efficient delay-driven approximate logic synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 11, pp. 3491–3504, Nov. 2023.
[5] C. Meng et al., "SEALS: Sensitivity-driven efficient approximate logic synthesis," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, 2022, pp. 439–444.
[6] G. Pasandi, M. Peterson, M. Herrera, S. Nazarian, and M. Pedram, "Deep-PowerX: A deep learning-based framework for low-power approximate logic synthesis," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design*, 2020, pp. 73–78.
[7] S. Hashemi, H. Tann, and S. Reda, "BLASYS: Approximate logic synthesis using Boolean matrix factorization," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2018, pp. 1–6.
[8] K. Balaskas, G. Zervakis, H. Amrouch, J. Henkel, and K. Siozios, "Automated design approximation to overcome circuit aging," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 11, pp. 4710–4721, Nov. 2021.
[9] K. Balaskas, F. Klemme, G. Zervakis, K. Siozios, H. Amrouch, and J. Henkel, "Variability-aware approximate circuit synthesis via genetic optimization," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 10, pp. 4141–4153, Oct. 2022.
[10] S. Salamin, G. Zervakis, O. Spantidi, I. Anagnostopoulos, J. Henkel, and H. Amrouch, "Reliability-aware quantization for anti-aging NPUs," in *Proc. IEEE/ACM Design, Autom. Test Europe (DATE)*, 2021, pp. 1460–1465.
[11] H. Wang et al., "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.
[12] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim, "RL-sizer: VLSI gate sizing for timing optimization using deep reinforcement learning," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2021, pp. 733–738.
[13] A. Agnesina, K. Chang, and S. K. Lim, "VLSI placement parameter optimization using deep reinforcement learning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–9.
[14] G. Pasandi, S. Nazarian, and M. Pedram, "Approximate logic synthesis: A reinforcement learning-based technology mapping approach," in *Proc. IEEE Int. Symp. Qual. Electron. Design (ISQED)*, 2019, pp. 26–32.
[15] J. Schlachter, V. Camus, K. V. Palem, and C. Enz, "Design and applications of approximate circuits by gate-level pruning," *IEEE Trans. Very Large Scale Integr. Syst. (TVLSI)*, vol. 25, no. 5, pp. 1694–1702, May 2017.
[16] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *Proc. IEEE/ACM Design, Autom. Test Eurpoe (DATE)*, 2013, pp. 1367–1372.
[17] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel, "Reliability-aware design to suppress aging," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2016, pp. 1–6.
[18] B. Tudor et al., "MOSRA: An efficient and versatile MOS aging modeling and reliability analysis solution for 45 nm and below," in *Proc. 10th IEEE Int. Conf. Solid-State Integr. Circuit Technol.*, 2010, pp. 1645–1647.
[19] A. B. Kahng, "New game, new goal posts: A recent history of timing closure," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2015, pp. 1–6.
[20] (Synopsys, Inc., Sunnyvale, CA, USA). *PrimeLib User Guide*. (2023). [Online]. Available: https://www.synopsys.com/implementation-and-signoff/signoff/primelib.html
[21] X. Wang, S. Tao, J. Zhu, Y. Shi, and W. Qian, "AccALS: Accelerating approximate logic synthesis by selection of multiple local approximate changes," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.
[22] K. Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014, *arXiv:1406.1078*.
[23] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2020, pp. 1725–1735.
[24] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.
[25] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 1928–1937.
[26] (EPFL, Lausanne, Switzerland). *The EPFL Combinational Benchmark Suite*. (2019). [Online]. Available: https://lsi.epfl.ch/page-102566-en-html/benchmarks/,
[27] (Synopsys, Inc., Sunnyvale, CA, USA). *Design Compiler User Guide*. (2023). [Online]. Available: https://www.synopsys.com/zh-cn/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html
[28] (Mentor Graphics Corporation, Wilsonville, OR, USA). *Modelsim User Guide*. (2023). [Online]. Available: https://eda.sw.siemens.com/en-US/ic/modelsim/.
[29] (Synopsys, Inc., Sunnyvale, CA, USA). *PrimeTime User Guide*. (2023). [Online]. Available: https://www.synopsys.com/cgi-bin/imp/pdfdla/pdfr1.cgi?file=primetime-wp.pdf
[30] D. Bertsekas, *Reinforcement Learning and Optimal Control*. Nashua, NH, USA: Athena Sci., 2019.

Yuyang Ye received the M.Sc. degree from The Hong Kong University of Science and Technology, Hong Kong, in 2020. He is currently pursing the Ph.D. degree with the National ASIC Research Center, Southeast University, Nanjing, China.

His current research interests include machine learning for EDA, timing analysis, and optimization.

Dr. Ye received the Best Student Paper Award from ICSICT 2022.

Tinghuan Chen (Member, IEEE) received the B.Eng. and M.Eng. degrees in electronics engineering from Southeast University, Nanjing, China, in 2014 and 2017, respectively, and the Ph.D. degree in computer science and engineering from The Chinese University of Hong Kong, Hong Kong, in 2021.

He is currently an Assistant Professor with the School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, China. His research interests include machine learning for EDA and deep learning accelerators.

Yifei Gao received the B.Eng. degree from Southeast University, Nanjing, China, in 2021, where he is currently pursing the M.Eng. degree with the National ASIC Research Center.

His current research interests include timing analysis and optimization.

Hao Yan (Member, IEEE) received the B.S. degree from the Dalian University of Technology, Dalian, China, in 2011, and the M.S. and Ph.D. degrees from Southeast University, Nanjing, China, in 2014 and 2018, respectively.

He is currently an Associate Professor with the National ASIC Research Center, Southeast University. His research focuses on design methodology for wide-voltage and high-efficiency design, including timing analysis and optimization.

Bei Yu (Senior Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Dr. Yu received ten Best Paper Awards from IEEE TSM 2022, DATE 2022, ICCAD 2021 and 2013, ASPDAC 2021 and 2012, ICTAI 2019, *Integration, the VLSI Journal* in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, and six ICCAD/ISPD contest awards. He has served as a TPC Chair for ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees. He is an Editor of IEEE TCCPS Newsletter.

Longxing Shi (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees from Southeast University, Nanjing, China, in 1984, 1987, and 1992, respectively.

From 1992 to 2000, he was an Associate Professor with the School of Electronic Science and Engineering, Southeast University, where he has been a Professor and the Dean of the National ASIC Research Center since 2001. He has authored one book and over 130 articles. His current research interest includes ultra-low power IC design and design methodology.