Annotating Slack Directly on Your Verilog: Fine-Grained RTL Timing Evaluation for Early Optimization

Wenji Fang
HKUST & HKUST(GZ)
wenjifang1@ust.hk

Shang Liu
HKUST
sliudx@connect.ust.hk

Hongce Zhang*
HKUST & HKUST(GZ)
hongcezh@ust.hk

Zhiyao Xie*
HKUST
eezhiyao@ust.hk

ABSTRACT

In digital IC design, compared with post-synthesis netlists or layouts, the early register-transfer level (RTL) stage offers greater optimization flexibility for both designers and EDA tools. However, timing information is typically unavailable at this early stage. Some recent machine learning (ML) solutions propose to predict the total negative slack (TNS) and worst negative slack (WNS) of an entire design at the RTL stage, but the fine-grained timing information of individual registers remains unavailable. In this work, we address the unique challenges of RTL timing prediction and introduce our solution named RTL-Timer. To the best of our knowledge, this is the first fine-grained general timing estimator applicable to any given design. RTL-Timer explores multiple promising RTL representations and proposes customized loss functions to capture the maximum arrival time at register endpoints. RTL-Timer's fine-grained predictions are further applied to guide optimization in a standard synthesis flow. The average results on unknown test designs demonstrate a correlation above 0.89, contributing around 3% WNS and 10% TNS improvement after optimization.

1 INTRODUCTION

Performance is a primary design objective in digital integrated circuit (IC) design. To achieve desired performance, huge engineering efforts are spent on the analysis and optimization of *timing*, which describes the maximum delays of signal propagation in IC. However, accurate static timing analysis (STA) tools require precise resistance and capacitance values as inputs, which are often unavailable until late post-layout or sign-off stages.

However, the sign-off stage is often too late to maximally optimize timing. Optimizations are generally preferred at the early stage and high-abstraction level, when many design decisions are not finalized yet. But such early optimization needs a good early timing evaluation to guide it, predicting the ultimate timing in advance. It is extremely challenging for traditional analytical STA tools [8] to predict timing at early design stages. For a netlist, due to the lack of wire length information, industry-standard STA tools fail to correlate well with ground-truth timing labels from layout [16]. As for the even earlier register-transfer level (RTL), existing STA tools do not even provide any guesses.

The RTL is a critical stage where designers define precise design behaviors with hardware description languages (HDLs) like Verilog, VHDL, or Chisel. Compared with post-synthesis netlists or layouts, the early RTL stage enables significantly higher optimization flexibility, maximally allowing designers or EDA tools to make fine-grained design decisions. However, timing evaluation is unavailable at this early stage and thus optimization lacks guidance.

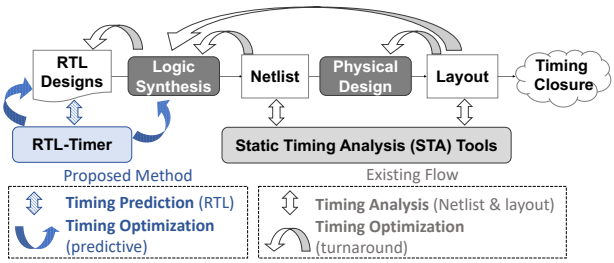


Figure 1: Design flow with our RTL-Timer, enabling RTL-stage timing evaluation for predictive optimizations.

In recent years, machine learning (ML) methods have been developed to provide early timing predictions. Explorations mostly target the layout [1, 2, 6, 7, 9, 15] and netlist [16] stages. But the more challenging early RTL stage is seldom explored until 2022, largely due to its essentially higher difficulty. We summarize two unique challenges of RTL-stage timing prediction below. They make most post-synthesis solutions [1, 2, 6, 7, 9, 15, 16] inapplicable:

(1) Design RTL is originally in HDL code format, which cannot be directly processed by either ML or traditional STA tools.
(2) There is no direct mapping between most RTL signals (i.e., model raw input) and post-synthesis cells/nets, where delay labels are supposed to be annotated.

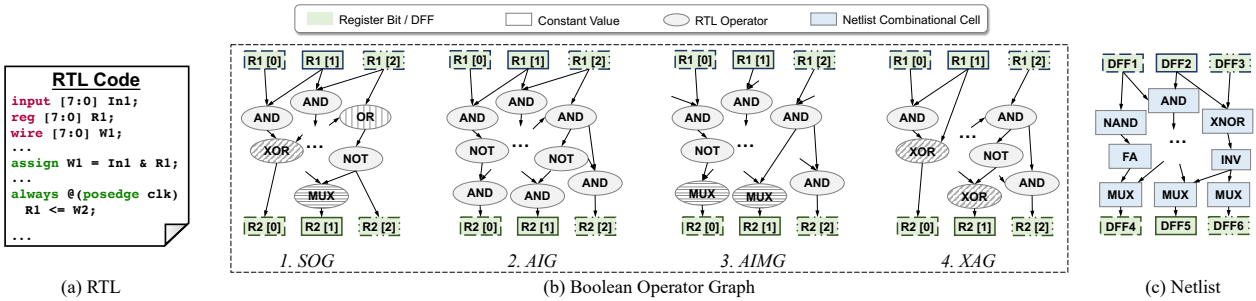
In this work, we tackle the above two challenges with our solution named RTL-Timer, whose position in the design flow is shown in Fig. 1. To the best of our knowledge, this is the first general fine-grained RTL-stage timing model that is applicable to any given new design. In addition to overall worst negative slack (WNS) and total negative slack (TNS), our fine-grained solution predicts the slack information of individual sequential RTL signals. This is a highly challenging task, but RTL-Timer is accurate enough to benefit optimization. We have further implemented automatic annotation of predicted fine-grained slack information on user-provided HDL code. We also demonstrated successful optimizations based on RTL-Timer's predicted fine-grained slack information.

RTL-Timer tackles the two aforementioned challenges with innovative techniques. 1) To handle the code format of design RTL, RTL-Timer systematically explores various RTL representations and proposes the ML-friendly ensemble of multiple representation candidates. 2) To address the mismatch between RTL signals (i.e., model raw inputs) and cells/nets with timing labels, we utilize the consistency in registers between RTL and netlist. It captures the maximum arrival time at each register endpoint by sampling multiple paths in its input logic. A customized loss function is developed to enable end-to-end model training. Also, instead of directly evaluating the signals, RTL-Timer focuses on each bit of the signal. Finally, the individual bits are aggregated back into the complete RTL signal, providing a comprehensive evaluation.

*Corresponding Author

DAC '24, June 23–27, 2024, San Francisco, CA

ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/https://doi.org/10.1145/3649329.3655671



(a) RTL (b) Boolean Operator Graph (c) Netlist

Figure 2: RTL representations explored in this work.

Methods	Fine-Grained	General Solution		Applied in Optimization
		Sequential	Cross-Design	
ICCAD'22 [10]			✓	
ISCA'22 [17]		✓	✓	
ICCAD'22 [13]		✓	✓	
ICCAD'23 [4]		✓	✓	
MLCAD'23 [12]			✓	
MLCAD'23 [11]	✓		✓	
MLCAD'23 [14]	✓	✓		✓*
RTL-Timer	✓	✓	✓	✓

* This work [14] only provides one manual optimization example on a piece of RTL code, without further detailed optimization results on a complete design.

Table 1: Existing RTL timing evaluators. The earliest exploration started in 2022. RTL-Timer is the first general fine-grained RTL timing estimator, which is also the first to demonstrate a positive impact in realistic optimizations.

Table 1 summarizes all existing explorations in RTL-stage timing prediction. Many works [10–12] only accept small-scale combinational designs, while some [14] cannot even be applied to unknown new designs. They are not general solutions that can apply to any design type. In addition, most methods [4, 10, 12, 13, 17] only target TNS or WNS values of a whole design. Also, most works perform prediction only, without being applied in realistic optimizations. Our contributions are summarized below.

- To the best of our knowledge, RTL-Timer[1] is the first fine-grained general timing estimator at the early RTL stage. It demonstrates $R = 0.89$ and ranking coverage = 80% in fine-grained slack values, and also achieves state-of-the-art accuracy in overall design TNS and WNS predictions.
- We advance the understanding of data-driven RTL code processing by exploring multiple promising ML-friendly representations and an ensemble of them.
- To handle the mismatch between RTL signals and netlist cells/nets, we propose a customized ML method to capture the maximum arrival time of each register endpoint. Both regression and learning-to-rank models are explored.
- Based on the fine-grained timing prediction, RTL-Timer further predicts overall TNS and WNS, achieving state-of-the-art $R = 0.98$ and $R = 0.91$, respectively.
- To demonstrate the effectiveness of RTL-Timer and its benefit in early optimizations, we apply it in two unprecedented applications: 1) We enabled automatic annotating slack prediction of sequential signals in RTL code; 2) We control optimization options `group_path` and `retime` during logic synthesis based on predictions. Experiments show an improvement up to 33.5% in TNS (avg. 9.9%) and 16.4% in WNS (avg. 3.1%) with negligible impact on power and area. This post-synthesis improvement remains significant after the layout.

[1]It is open-sourced in https://github.com/hkust-zhiyao/RTL-Timer

2 PROBLEM FORMULATION

We denote an initial design HDL code (e.g., Verilog) as $\mathcal{V}$, and its post-synthesis gate-level netlist as $\mathcal{N}$. Each register as the timing path endpoint[2] in the design is denoted as ep_i. Because of the consistency between RTL sequential signals and netlist registers, ep_i appears in both the RTL design $\mathcal{V}$ and the netlist $\mathcal{N}$. Our framework F will predict the post-synthesis endpoint arrival time[3] ($AT_{\mathcal{N}}$) and the associated ranking ($Rank_{\mathcal{N}}$), as formulated below:

Problem 1 (Register arrival time value prediction).

$$\forall ep_i \in \mathcal{V}, F_{AT}(ep_i) \rightarrow AT_{\mathcal{N}}(ep_i) \tag{1}$$

Note that accurate arrival time prediction at the RTL stage is extremely challenging — even robust regression models may lead to significant ranking variances. Therefore, we strategically reframe the above as a ranking problem:

Problem 2 (Register arrival time ranking prediction).

$$\forall ep_i \in \mathcal{V}, F_{Rank}(ep_i) \rightarrow Rank_{\mathcal{N}}(ep_i) \tag{2}$$

3 METHODOLOGY

3.1 Universal ML-friendly RTL Representation

The first challenge in RTL prediction is that the initial HDL code format cannot be directly processed by the STA tool or ML models. Existing RTL representations, such as Binary Decision Diagrams (BDD), Conjunctive Normal Form (CNF), and And-Inverter Graphs (AIGER), focus primarily on logic transformations for logic synthesis and verification. They are not optimized for ML-based solutions, which require exposing the correlation between RTL and netlist. Existing works have adopted signal-level representations like abstract syntax tree (AST) [13, 17] and bit-level representations like simple-operator graph (SOG) [4]. These representations are adopted as ad hoc solutions, without systematically exploring better candidates.

In this work, we propose a versatile ML-friendly RTL representation, named Boolean operator graph (BOG), denoted as $\mathcal{R}$ and illustrated in Fig. 2(b). BOG is a universal bit-level RTL representation and can be specialized into concrete variants (SOG, AIG, etc.) by selecting different Boolean operators (AND, NOT, OR, XOR, MUX). Besides viewing BOG as a graph of registers and operators, we also treat $\mathcal{R}$ as a pseudo netlist, where registers and operators are viewed as pseudo standard cells from the liberty file.

This bit-level BOG enforces the one-to-one mapping between sequential RTL signal bits and bit-wise netlist registers. It provides the basis for the RTL-stage fine-grained endpoint modeling. RTL-Timer employs four distinct BOG representations: SOG, AIG, AIMG, and XAG. These four representations share the same functionality for

[2]A tiny portion of endpoints are primary output (PO) pins.

[3]We assume a fixed clock frequency, implying slack is solely determined by arrival time.

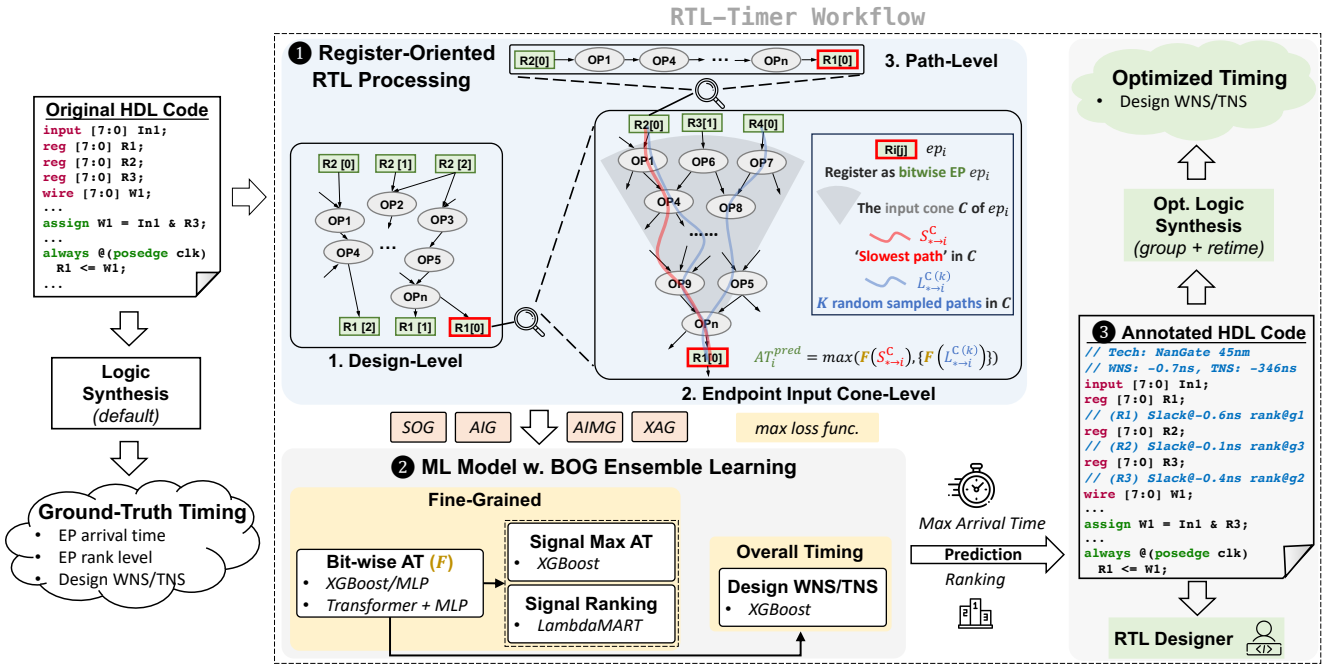


Figure 3: RTL-Timer workflow. We use register-oriented RTL processing and ensemble four RTL representations, enabling fine-grained and overall timing modeling. Predictions are annotated on HDL files, aiding designers and enhancing logic synthesis.

each design, offering rich and multidimensional analytical perspectives. Intuitively, AIG, with only basic AND and NOT operators, provides fundamental insights. In contrast, SOG, consisting of a broader range of operator types, is more similar to the target netlist. AIMG and XAG reside at the intermediate levels between SOG and AIG.

Furthermore, we employ ensemble learning to aggregate the strengths of these four representations. This fusion contributes to a more accurate and robust timing modeling approach with significantly reduced variance across various designs. Detailed experimental results will be presented in Section 4.

3.2 A General Register-Oriented RTL Processing Workflow

While leveraging the BOG representation $\mathcal{R}$, we face another challenge in RTL prediction: the signals in $\mathcal{R}$ cannot match the cells/nets in the netlist $\mathcal{V}$, leading to a lack of fine-grained labels for $\mathcal{R}$. Fortunately, the register consistency in BOG enables us to label each bit-wise register (i.e., endpoint ep_i) with the slack from $\mathcal{N}$ as reported by STA. Nonetheless, a mismatch remains for internal operators.

Inspired by the propagation mechanism in STA – where each ep_i accumulates arrival time from all its driving registers, and uses the maximum arrival time to compute the slack – we propose a comprehensive register endpoint-oriented RTL processing approach. This method aims to capture the timing-related pattern of internal operators, as demonstrated in step ❶ of Fig. 3.

If we assume the netlist $\mathcal{N}$ and representation $\mathcal{R}$ were exactly the same, the arrival time of each endpoint ep_i in $\mathcal{N}$ will simply only depend on the slowest path $S^{\mathcal{R}}_{*\rightarrow i}$ from $\mathcal{R}$. But logic optimization and technology mapping will optimize $\mathcal{R}$ towards $\mathcal{N}$ in logic synthesis. Therefore, other paths ending at ep_i in $\mathcal{R}$ may also contribute to the ultimate slowest path in $\mathcal{N}$, and we also need to consider these paths, though they may not be the slowest in $\mathcal{R}$.

Specifically, we backtrack from each ep_i to all driving registers in $\mathcal{R}$ to obtain its input cone C, which includes all input logic of ep_i. We then sample two different types of paths from C: 1) The slowest path in $\mathcal{R}$: Since we construct $\mathcal{R}$ as a pseudo netlist, we can efficiently traverse $\mathcal{R}$ in topological order and perform the traditional STA algorithm on it. Applying this efficient pseudo-STA process directly on $\mathcal{R}$, we can trace the "slowest path" $S^{\mathcal{R}}_{*\rightarrow i}$ ending at each register ep_i. 2) Random paths in $\mathcal{R}$: We also randomly sample other paths in C, denoted as $\{L^{\mathcal{R}(k)}_{*\rightarrow i} | k \in [1, K_i]\}$, where the sample number K_i is proportional to the number of driving registers.

Given a path-level model F_{AT}, the ultimate bit-wise max arrival time prediction of ep_i depends on the maximum prediction of all paths ending at ep_i, as formulated below:

$$\begin{aligned} AT_i^{predict} &= max(F_{AT}(S^{\mathcal{R}}_{*\rightarrow i}), \{F_{AT}(L^{\mathcal{R}(k)}_{*\rightarrow i})\}), \\ Loss_i &= LossFunc(AT_i^{predict}, AT_i^{label}). \end{aligned} \tag{3}$$

This loss function is differentiable with respect to model F_{AT}, enabling end-to-end gradient-based model training. In this way, we solve the lack of labels resulting from the gap between RTL and netlist and establish a crucial link between the internal operators in $\mathcal{R}$ and the target endpoints' max arrival time.

Building upon bit-wise endpoints, we can calculate the timing for signal-wise endpoints, where the signals are the variables originally defined in the HDL code $\mathcal{V}$. Since an RTL signal can comprise multiple bits, we determine the arrival time of each signal-wise endpoint based on the longest arrival time among all its bits, referred to as the signal max arrival time in subsequent discussions. As demonstrated in Section 2, we address both the signal max arrival time value regression and the critical level ranking tasks.

In addition to fine-grained endpoints, we also target the overall timing metrics for the whole design (i.e., TNS and WNS), which are directly calculated utilizing the negative register slack.

3.3 Feature Exploration for RTL Processing

We extract three levels of features during our RTL processing, as listed in Table 2. 1) Design-level: Global design features are crucial for comparing endpoints across different designs. Even similar

synthesized timing paths from distinct designs can show varied slacks, often due to diverse optimization efforts in logic synthesis. Factors like design size and the critical ranking of timing paths are thus included as our design features. 2) Cone-level: The number of driving registers is used to evaluate the size of the cone. It also helps to differentiate the similar endpoints. 3) Path-level: For timing paths identified by the STA tool, we extract physical-related features and compute key statistics like sum, average, and standard deviation. There is a reasonable correlation R between each feature and the arrival time label at endpoints, providing insightful patterns for further fine-grained model (i.e., F_{AT}).

Our ensemble approach incorporates the four proposed BOG representations for robust timing modeling. It combines the predictions based on the four BOGs and is supplemented by statistics such as the maximum, minimum, and average of these predictions.

Type	Feature	Avg. R
Design	Rank level % of the endpoint rank # of sequential cells # of combinational cells # of total cells	/
Cone	# driving reg of input cone	0.45
Path	Arrival time by STA on $\mathcal{R}$	0.43
	# of level of the timing path	0.51
	# of operators	0.56
	Fanout	0.40
	Load capacitance	0.38
	Slew	0.38

Table 2: Feature summary.

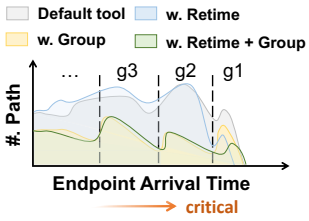


Figure 4: Optimization options in logic synthesis.

3.4 ML Model Exploration for RTL Processing

As for the model, we propose to explore multiple promising ML models to evaluate both the fine-grained timing and TNS/WNS based on our processed RTL representations, as shown in step ❷ of Fig. 3.

3.4.1 Bit-wise Endpoint Modeling. We investigate various ML models, each integrated with the customized loss function tailored for register-oriented RTL processing. The models include: 1) Tree-based: A lightweight XGBoost model; 2) MLP: A multilayer perceptron (MLP) model; 3) Transformer: This model combines a transformer, for local path modeling, with an MLP to capture global features. Once the modeling is complete, we can predict the max arrival time at all bit-wise endpoints within a design, and then further calculate their ranking.

3.4.2 Signal-wise Endpoint Modeling. As previously mentioned, the signal max arrival time is determined by the bit with the longest arrival time. Our signal model is thus constructed leveraging the bit-level predictions. For the regression model, we employ a lightweight tree-based model. Regarding the ranking model, we reframe the problem as a learning-to-rank (LTR) task [3]. Unlike regression methods that predict absolute target values, LTR learns and predicts the relative ranking among items. It uses supervised learning with training data comprising queries, each containing a group of documents with features and relevance scores as labels. In our context, each design is treated as a query, all its endpoints are documents, and their critical ranking levels are the labels. The ranking model orders the critical levels of endpoints within each design. We leverage a pair-wise ranking model to capture feature distinctions between timing path pairs.

3.4.3 Design Overall Timing Modeling. Given that TNS and WNS rely on the negative register slack, our proficiency in accurate fine-grained endpoint modeling leads to high prediction accuracy. In our model, we calculate TNS and WNS using the bit-wise predictions for ensemble features. Additionally, design-level features are incorporated to distinguish among designs of varying scales. Another tree-based model is employed for this regression task.

3.5 Optimization Enabled by RTL-Timer

The fine-grained predictions are unprecedentedly applied in two early optimization applications: 1) For manual optimization, RTL-Timer provides early feedback to RTL designers by directly annotating detailed timing information on HDL; 2) For automatic optimization, RTL-Timer can set fine-grained optimization options in the synthesis script. This is illustrated in step ❸ of Fig. 3.

3.5.1 Automatic Slack Annotation on HDL. We have implemented an annotation tool that automatically applies RTL-Timer's timing evaluation onto the original HDL code. It marks the technology node and the overall TNS/WNS for the whole design. For each sequential signal, it annotates the predicted slack value and its relative ranking group. This tool may become a plug-in for an Integrated Development Environment (IDE), assisting RTL designers to modify timing-critical components without the logic synthesis process.

3.5.2 Enhancing Logic Synthesis Process. RTL-Timer can efficiently control optimization options during logic synthesis. Here, we highlight two key options supported by commercial tools: path grouping and register retiming. By combining these two approaches together, we efficiently improve both TNS and WNS. Fig.4 illustrates their effects on the arrival time distribution.

Default logic synthesis tools only focus on the most critical timing violations, often leaving huge space for improvement in other timing endpoints. To address this, we divide all endpoints into four groups based on their predicted signal-wise endpoint rankings. We then apply the `group_path` command for each register signal, allocating specific optimization efforts to each group. This strategy improves TNS without affecting WNS. As for retiming, which is not activated by default due to the lack of proper guidance, we focus on the top 5% of critical endpoints. Utilizing the `retime` command, registers are repositioned across combinational logic gates for more balanced timing results, particularly beneficial for WNS optimization.

Note that the specific register names need to be assigned to the above two commands, which was impossible without our fine-grained predictions.

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

We implement our models using Scikit-learn and Pytorch frameworks. Regarding the model hyper-parameters, all the XGBoost models across different granularities are constructed with 100 estimation trees and a maximum depth of 45. The MLPs are configured with 3 layers and a hidden dimension of 512. The transformer model shares the same hyper-parameters as used in [4, 17]. Since there is no existing work on the general fine-grained timing prediction for sequential RTL designs, we adopt the SOTA layout-stage solution from [15] as a baseline, where we customize a GNN model to capture the bit-wise endpoint timing information. For the Learning-to-Rank task, we employ the pairwise LambdaMART algorithm with 100 estimators and a maximum depth of 30.

We train and evaluate our model on 21 open-source RTL designs using 10-fold cross-validation. Training and test datasets include strictly different designs. The benchmark spans various mainstream HDLs, covering a wide design scale range from 6K to 510K gates, as detailed in Table 3. For dataset generation, we utilize Synopsys Design Compiler and Cadence Innovus with the NanGate 45nm PDK, for logic synthesis and physical design, respectively. Static timing analysis is performed using Synopsys Prime Time.

Benchmarks*	#Designs	Design Size Range		HDL Type
		#K Gates	#K Endpoints	
ITC'99	6	9 - 45	0.4 - 1.3	VHDL
OpenCores	4	6 - 56	0.2 - 3.8	Verilog
Chipyard	3	20 - 32	2.5 - 4.1	Chisel
VexRiscv	8	7 - 510	1.2 - 146	SpinalHDL

* Small designs (<5K Gates) and those dominated by huge memory modules are excluded from the original benchmarks.

Table 3: Benchmark design information.

4.2 Evaluation Metrics

To evaluate solutions, we employ multiple metrics: the correlation coefficient (R), determination coefficient (R^2), mean absolute percentage error (MAPE), and critical level ranking coverage (COVR):

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y_i}|}{y_i}\times 100\%,\ \ \text{COVR} = \frac{1}{m}\sum_{g=1}^{m}\frac{\#(S_g \cap \hat{S_g})}{\#S_g}\times 100\%$$

The $\hat{y}$ and y are predictions and labels. In COVR, S_g is the group set categorized by the critical ranking level. In each design, we divide all the signal-wise endpoints into 4 groups: the top 5% as group1, 5%-40% as group2, 40%-70% as group3, and the remaining as group4. Notably, the groups are directly used in the subsequent optimizations. For each group, we calculate the coverage by dividing the number of prediction-label intersections by the label group size.

A higher R or R^2 and lower MAPE indicate better regression accuracy, and the higher coverage COVR represents a more accurate clustering of endpoints into the criticality groups.

4.3 Modeling Performance

As shown in Table 4, leveraging the fine-grained bit-wise prediction, which has a correlation of 0.88, our signal-wise prediction achieves a correlation of 0.89 and an 80% ranking coverage. The lightweight tree-based model outperforms all the deep learning models. We attribute this to our comprehensive feature engineering approach,

Fine-Grained	Method	R	MAPE (%)	COVR (%)
Bit-wise	Tree-based w/o sample	0.80	26	59
	MLP	0.71	35	56
	MLP w/o sample	0.65	38	54
	Transformer	0.73	35	57
	Customized GNN	0.25	53	46
	RTL-Timer	0.88	12	66
Signal-wise	Regression w/o bit-wise	0.56	28	56
	Ranking w/o bit-wise	/	/	39
	RTL-Timer (regression)	0.89	15	71
	RTL-Timer (ranking)	/	/	80
Overall	**Method**	R	R^2	MAPE (%)
WNS	SNS [17]	0.73	0.58	33
	MasterRTL [4]	0.89	0.74	15
	RTL-Timer	0.91	0.86	12
TNS	ICCAD'22 [13]	0.65	0.32	42
	MasterRTL [4]	0.96	0.94	34
	RTL-Timer	0.98	0.97	18

Table 4: Modeling accuracy comparison and ablation study.

	Metrics	SOG	AIG	AIMG	XAG	Ensemble
Bit-wise	Avg. R	0.85	0.75	0.76	0.77	**0.88**
	Std. R	0.18	0.25	0.26	0.21	**0.08**
Signal-wise	Avg. R	0.82	0.81	0.84	0.8	**0.89**
	Std. R	0.15	0.22	0.1	0.1	**0.06**
	Avg. COVR	65	71	72	71	**80**
	Std. COVR	18	19	21	21	**8**

Table 5: Comparison of four representation variants and the effect of ensemble learning to reduce variance.

which treats the original RTL design as tabular data — a domain where tree-based methods excel [5]. Additionally, building upon our bit-wise predictions, the overall design TNS and WNS show superior correlations of 0.98 and 0.91, respectively, which outperform all three SOTA methods [4, 13, 17].

Furthermore, to evaluate the contribution of each strategy of RTL-Timer in fine-grained modelings, we conduct ablation studies by selectively removing key strategies of our solution, also demonstrated in Table 4: 1) No sampled paths: By exploiting various models with only the slowest path, we observe notable accuracy decreases in all types of models (e.g., R drops 0.08 in the tree-based model). 2) Removing bit-wise prediction: If we eliminate the detailed analysis at the bit level, and directly model the RTL signal, there will be a significant decrease in accuracy for both regression (i.e., R: from 0.89 to 0.56) and ranking (i.e., COVR: from 80% to 39%) tasks. 3) Disabling the LTR method: Without the LTR model, the ranking accuracy noticeably diminishes, falling from 80% to 71%.

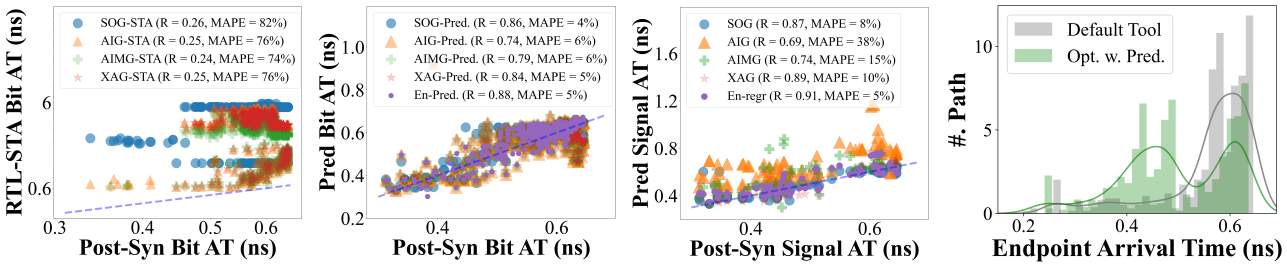
We also evaluate the four BOG variants and the impact of ensemble learning, as detailed in Table 5. The results reveal that each representation contributes to the prediction capabilities, with ensemble learning substantially reducing variance, thus ensuring robustness across diverse benchmarks and tasks.

Then we further look into *why ensemble representations are effective*. Analysis of feature importance shows that the average across the four representations forms the core of the final predictions, and omitting any representation leads to a decrease in accuracy. Notably, SOG and AIG carry more weight, reflecting their significant representational differences, whereas AIMG and XAG contribute similarly. Moreover, the inclusion of cone and design features markedly enhances the model's generalization capability across various designs during the representation ensemble.

Fig. 5 visualizes the experimental results for a design example b18_1. In Fig. 5(a), the pseudo-STA results of the four representations are shown. While these results do not closely match the post-synthesis arrival time, they offer valuable patterns for further modeling. Utilizing our ML models and ensemble learning technique (denoted as 'En'), both bit-wise and signal-wise predictions achieve high accuracy, as depicted in Fig. 5(b) and (c).

4.4 Optimization Performance

Table 6 demonstrates the automatic optimization results for each design. Leveraging the predicted rankings, our method effectively improves timing on both TNS and WNS for most designs, while maintaining or even decreasing the other design quality metrics (i.e., power and area). Cases where TNS or WNS worsen are considered non-optimized. Practically, designers can concurrently run default and optimization synthesis flows at the same time. In this way, solutions with better outcomes can be selected without time-consuming iterations. 'Avg1' considers all the results from the optimization flow, while 'Avg2' incorporates default synthesis results for those



(a) RTL-STA (b) Bit-wise prediction (c) Signal-wise prediction (d) Optimized distribution

Figure 5: Evaluated/predicted arrival time and optimized distribution for a design example b18_1.

non-optimized cases. Experiments illustrate an improvement up to 33.5% in TNS (avg. 9.9%) and 16.4% in WNS (avg. 3.1%). Notably, compared to the optimization with ground-truth ranking, our approach delivers comparable or superior results.

Although demonstrating improvements initially in the logic synthesis phase, our optimizations' impact remains significant throughout the placement stages. On average, we observe a 4.6% improvement in WNS and 6% in TNS after placement. These improvements even persist after the post-placement timing optimization, showing an average enhancement of 3.1% in WNS and 6.8% in TNS.

Fig. 5(d) showcases the improved arrival time distribution enabled by the RTL-Timer's predictions. Combining the two optimization options, the original high peak distribution is optimized into two lower peaks with better TNS. Meanwhile, the slowest arrival time (i.e., WNS) is effectively enhanced.

4.5 Runtime Analysis

RTL-Timer delivers fast fine-grained timing evaluation, without the need for the logic synthesis process. Overall, the modeling method consumes about 4% of the default synthesis runtime. It comprises two key parts: 1) RTL process. This involves converting HDL files into BOG variants, a process that can be parallelized. We measure its overhead based on the most time-consuming AIG construction (i.e., 3.2%). Additionally, the register-oriented RTL processing accounts for 0.9%. 2) Model inference time. It requires less than 0.1 seconds.

When employing our optimization in logic synthesis, the runtime extends by an average of 45%, due to the separate optimization efforts for different path groups and retiming.

5 CONCLUSION AND FUTURE WORK

In this paper, we present RTL-Timer, the first fine-grained general timing estimator for RTL designs, incorporating four RTL representations with a customized loss function to accurately evaluate the arrival time on registers. RTL-Timer facilitates optimizations for both designers and EDA tools. Our future work will focus on enhancing prediction accuracy for more detailed optimization options and explore the potential of automating RTL design optimization using the large language model (LLM).

6 ACKNOWLEDGMENTS

This work is partially supported by Hong Kong Research Grants Council (RGC) ECS Grant 26208723, National Natural Science Foundation of China (92364102, 62304192), ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR, and Guangzhou Municipal Science and Technology Project (Grant No.2023A03J0013).

Design	Singal-wise Pred. R	MAPE	COVR	Opt. w. Pred. (%) WNS	TNS	Pwr	Area	Opt. w. Real (%) WNS	TNS	Pwr	Area
syscdes	0.94	26%	94%	-1.3	-17	-0.8	1.8	-1.8	-17.3	0	2.6
syscaes	0.86	23%	77%	-1.3	-13.7	2.1	3.2	-0.1	-14.4	2.1	3.5
Vex_1	0.87	24%	86%	6.9	7	-3.1	-2.8	-0.3	-1.1	0.5	-0.8
b20*	0.91	7%	86%	5.6	-4.3	26.2	25	0.2	-6.6	26.2	23.4
Vex_2	0.86	16%	83%	-0.2	-1.6	-0.9	0	-0.7	-1.8	-0.6	0.3
Vex_3	0.93	30%	86%	-2.8	-4.8	3.9	1.2	-0.1	-2.2	1.9	1
b22*	0.74	18%	83%	0.7	-4.8	23.4	23.3	2.9	0.4	22.7	20.2
b17	0.93	8%	75%	1.9	-5.2	2.2	0	-0.9	-5.6	0.2	1
b17_1	0.94	5%	79%	5.8	-3.2	1.7	2.1	0.9	-5.8	-0.6	0.4
Rocket1	0.89	11%	63%	-7.1	-21.4	2.8	1.7	-3.7	-25.4	-66	2.9
Rocket2	0.92	18%	64%	-7	-23.1	-69.4	-0.6	-4.1	-23.1	1.6	-0.8
Rocket3	0.88	12%	69%	-7.2	-17.4	-69.4	-0.6	-6.2	-18.3	-69.7	0.8
conmax	0.91	12%	83%	-1.9	-3	3.4	2.7	-0.9	-0.6	3	2.1
b18	0.82	13%	84%	-16.4	-33.5	3.8	3.9	-17.9	-35.5	3.6	3.4
b18_1	0.88	10%	86%	-3.9	-26	0.5	-0.4	-9.8	-27	1.8	0.2
FPU	0.89	31%	85%	6.2	0.2	0.5	1	4.3	-1.7	-86.6	0.6
Marax	0.88	16%	78%	-1.7	-3	-0.1	-0.1	2.4	-2.2	0	-0.5
Vex_4	0.79	16%	67%	-4.8	-18.6	0	-0.7	-7.2	-21.9	0.4	-1
Vex5	0.92	4%	81%	-1.8	-4.6	0.2	-1.3	-3.6	-10.7	0.3	-0.6
Vex6	0.94	6%	82%	-1.5	-12.8	0.4	-0.5	-3	-5.3	0.3	-1.3
Vex7	0.87	6%	81%	-5.7	-7.2	0.5	-0.8	-3	-12.3	0.5	-0.2
Avg1	**0.89**	**15**	**80**	**-1.9**	**-10.4**	**-3.4**	**2.8**	**-2.5**	**-11.2**	**-7.5**	**2.7**
Avg2				**-3.1**	**-9.9**	**-5.9**	**0.5**	**-3**	**-10.6**	**-5.7**	**0.6**

* Special cases with small size (<15K) and low sequential cell ratio, which are hard to further optimize, resulting in huge power and area overhead.

Table 6: Optimization enabled by predictions and labels.

REFERENCES

[1] Erick Carvajal Barboza, Nishchal Shukla, Yiran Chen, et al. 2019. Machine learning-based pre-routing timing prediction with reduced pessimism. In *DAC*.
[2] Peng Cao, Guoqing He, and Tai Yang. 2022. TF-Predictor: Transformer-based Pre-Routing Path Delay Prediction Framework. *IEEE TCAD* (2022).
[3] Zhe Cao, Tao Qin, Tie-Yan Liu, et al. 2007. Learning to rank: from pairwise approach to listwise approach. In *ICML*.
[4] Wenji Fang, Yao Lu, Shang Liu, et al. 2023. MasterRTL: A Pre-Synthesis PPA Estimation Framework for Any RTL Design. In *ICCAD*.
[5] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data?. In *NeurIPS*.
[6] Zizheng Guo, Mingjie Liu, Jiaqi Gu, et al. 2022. A timing engine inspired graph neural network model for pre-routing slack prediction. In *DAC*.
[7] Xu He, Zhiyong Fu, Yao Wang, et al. 2022. Accurate timing prediction at placement stage with look-ahead RC network. In *DAC*.
[8] Tsung-Wei Huang and Martin DF Wong. 2015. OpenTimer: A high-performance timing analysis tool. In *ICCAD*.
[9] Andrew B Kahng, Uday Mallappa, and Lawrence Saul. 2018. Using machine learning to predict path-based slack from graph-based timing analysis. In *ICCD*.
[10] Daniela Sánchez Lopera and Wolfgang Ecker. 2022. Applying GNNs to Timing Estimation at RTL. In *ICCAD*.
[11] Daniela Sánchez Lopera, Ishwor Subedi, et al. 2023. Using Graph Neural Networks for Timing Estimations of RTL Intermediate Representations. In *MLCAD*.
[12] Yikang Ouyang, Sicheng Li, Dongsheng Zuo, et al. 2023. ASAP: Accurate Synthesis Analysis and Prediction with Multi-Task Learning. In *MLCAD*.
[13] Prianka Sengupta, Aakash Tyagi, Yiran Chen, et al. 2022. How Good Is Your Verilog RTL Code? A Quick Answer from Machine Learning. In *ICCAD*.
[14] Prianka Sengupta, Aakash Tyagi, Yiran Chen, et al. 2023. Early Identification of Timing Critical RTL Components using ML based Path Delay Prediction. In *MLCAD*.
[15] Ziyi Wang, Siting Liu, Yuan Pu, et al. 2023. Restructure-Tolerant Timing Prediction via Multimodal Fusion. In *DAC*.
[16] Zhiyao Xie, Rongjian Liang, Xiaoqing Xu, et al. 2022. Preplacement net length and timing estimation by customized graph neural network. *IEEE TCAD* (2022).
[17] Ceyu Xu, Chris Kjellqvist, and Lisa Wu Wills. 2022. SNS's not a synthesizer: a deep-learning-based synthesis predictor. In *ISCA*.