

ParaGate: Parasitic-Driven Domain Adaptation Transfer Learning for Netlist Performance Prediction

Bin Sun^{1,5}, Jingyi Zhou², Jianan Mu^{*1}, Zhiteng Chao¹, Tianmeng Yang³, Ziyue Xu⁴, Jing Ye¹, Huawei Li^{*1}

¹*State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences*

²*Department of Electronic Engineering, Tsinghua University*

³*Peking University*

⁴*School of Information Engineering, Minzu University of China*

⁵*University of Chinese Academy of Sciences*

sunbin23@mails.ucas.ac.cn, zhou-jy22@mails.tsinghua.edu.cn,

{mujianan, chaozhiteng20g, yejing, lihuawei}@ict.ac.cn,

youngtimmy@pku.edu.cn, 22012820@muc.edu.cn

Abstract—In traditional EDA flows, layout-level performance metrics are only obtainable after placement and routing, hindering global optimization at earlier stages. Although some neural-network-based solutions predict layout-level performance directly from netlists, they often face generalization challenges due to the black-box heuristics of commercial placement-and-routing tools, which create disparate data across designs. To this end, we propose ParaGate, a three-step cross-stage prediction framework that infers layout-level timing and power from netlists. First, we propose a two-phase transfer-learning approach to predict parasitic parameters, pre-training on mid-scale circuits and fine-tuning on larger ones to capture extreme conditions. Next, we rely on EDA tools for timing analysis, offloading the long-path numerical reasoning. Finally, ParaGate performs global calibration using subgraph features. Experiments show that ParaGate achieves strong generalization with minimal fine-tuning data: on openE906, its arrival-time R^2 from 0.119 to 0.897. These results demonstrate that ParaGate could provide guidance for global optimization in the synthesis and placement stages.

I. INTRODUCTION

As circuit designs scale up, the stage-by-stage approach in traditional EDA flows often leads to slow pipeline-based feedback for final performance metrics, since obtaining timing, power, and other indicators requires completing the entire synthesis and physical design cycle, thus hindering global optimizations [1]–[4]. Therefore, AI-assisted cross-stage performance prediction has emerged, aiming to provide earlier and more effective guidance. Some recent studies have trained graph neural networks (GNNs) on RTL [5], [6] or netlists [7]–[12] to model circuit structures and signal propagation, thereby predicting post-layout metrics such as timing and power with strong accuracy on standard benchmarks.

However, one of the limitations of existing solutions lies in their generalizability [8], [13]. As shown in Fig. 1(a), we replicated a state-of-the-art GNN-based end-to-end predictor and found that when the test circuit differs significantly from the circuits in the training set (e.g., a larger-scale circuit), its predictive performance worsens. Some prior studies have achieved favorable performance within known designs by predicting remaining parts based on partial knowledge from the same circuit, or by learning from similar circuits [14], [15]. However, there are scenarios where the user must predict the

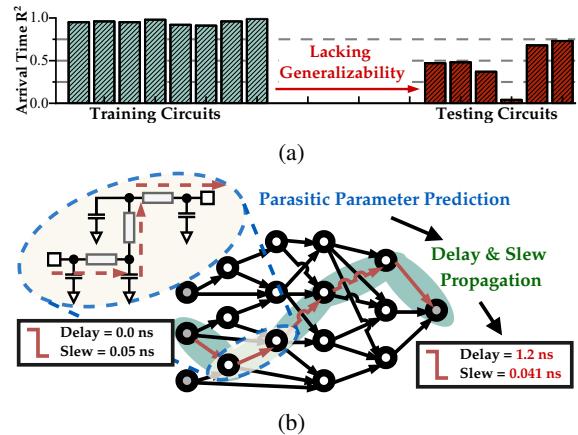


Fig. 1 (a) End-to-end method lacking generalizability; (b) Potential task coupling in PPA prediction

PPA of an entirely new circuit. A straightforward approach is to build a training set encompassing numerous diverse circuits; yet, the topological design space of functional units is practically infinite, making it impossible to exhaustively represent all structures. Therefore, beyond simply expanding the training set, it is also necessary to optimize the prediction methodology.

Predicting layout-level PPA from the netlist is fundamentally physical numerical reasoning based on connectivity relationships. As shown in Fig. 1(b), it in fact encompasses two tasks: Task (1): Predict the parasitic-parameter topology. For this task, parasitic parameters are primarily determined by the wires between two pins, so the model mainly needs to learn the correspondence between circuit structures and parameters within a smaller local region. Task (2): Numerical reasoning for arrival time and power based on the topological graph to emulate delay and slew propagation. However, neural networks are not well-suited for numerical reasoning along long topological paths [16], [17]. Moreover, the path topologies of different circuits often differ significantly, further increasing the difficulty of generalization. In summary, for cross-stage prediction, the significant structural differences in long paths among various cir-

cuits—together with the unfriendliness of path-topology-based numerical reasoning to neural networks—make it challenging to train a generalized model.

Based on this analysis, a concise and seemingly promising framework appears to emerge: decoupling end-to-end cross-stage PPA prediction into two steps. The first step is to use a neural network to predict the physical parameters between pins—essentially providing a cross-stage “forecast” for the layout level. The second step is to leverage the mature timing reasoning engine in EDA tools to obtain the timing and power metrics. However, although this decoupling seems promising, achieving cross-stage parasitic parameter prediction for general circuits is still far from trivial. Some studies predict parasitic parameters at the netlist level, but they mainly focus on specific circuit structures, such as SRAM [14]. Developing a generalized model capable of predicting across different circuits is more challenging. First, in large-scale digital circuits, the range of parasitic parameters between nodes can span four to five orders of magnitude, leading to data imbalance. Second, commercial placement-and-routing tools operate as black boxes using heuristic algorithms for optimization, which work uniquely across different circuit structures and paths. These issues pose challenges for the generalization of prediction models. In addition, when modeling the routing, one cannot solely capture the cell information at both endpoints of the wires; it is necessary to incorporate subgraph-level structure to perceive vital information like congestion.

To this end, we propose ParaGate, a cross-stage prediction framework. It takes the netlist as input and predicts the layout-level timing and power metrics. ParaGate is a three-step predictive framework. At its core is step 1, a two-phase transfer learning framework dedicated to parasitic parameter prediction. We first pre-train on small- and medium-scale circuits to learn general placement-and-routing rules; then we fine-tune the model on data from large-scale circuits with more complex conditions, sampling to capture extreme cases with unique characteristics. In step 2, we employ EDA tools to perform timing analysis. Finally, in step 3, we use a model that incorporates global information to calibrate the results. Our contributions are summarized as follows:

- We propose a three-step framework: using an AI model to predict physical parameters, invoking EDA tools to infer metrics on long paths, and finally performing error correction by incorporating subgraph information.
- For physical parasitic parameter prediction, we innovatively propose a “pre-training + fine-tuning” training framework to better capture general patterns and extreme-value distributions.
- We also employ a calibration model that captures subgraph features to further refine the prediction metrics.
- Experimental results confirm that ParaGate achieves outstanding generalization on unseen samples with only 20% target domain data for fine-tuning. This is evident in the *openE906* arrival time R^2 improving significantly from 0.119 to 0.897, and a total power relative error of just 0.909% on *BoomTile_Tiny*.

II. PRELIMINARY

A. PPA Analysis and Cross-Stage Prediction Models

Current research on performance, power, and area (PPA) analysis and prediction at the gate-level netlist can be broadly categorized into two types: The first category focuses on accelerating time-consuming analysis tasks within pre-layout netlists by leveraging machine learning methods. The second category primarily concentrates on cross-stage prediction. Among these, representative works include Timer-inspired GNN [8], which draws inspiration from timing analysis tools to realize an end-to-end pre-routing slack prediction framework. LaRC-Timer [7] performs delay prediction by rapidly estimating an RC Network and subsequently applying a tree model. Wang *et al.* [11] addresses the impact of topological changes in the netlist caused by back-end restructuring on delay prediction; Lyu *et al.* [12] proposed an interpretable and layout-aware end-to-end timing prediction method; PreRoutGNN [10] proposes a two-step pre-routing timing prediction scheme based on graph autoencoders; and Du *et al.* introduced PowPredict [9], the first cross-stage power prediction model with generalization capabilities. It is noteworthy that current efforts in cross-stage prediction are predominantly based on post-layout information, primarily tackling pre-routing timing and power prediction.

Moreover, in addressing physical information discrepancies, some researchers have pursued alternative avenues: they attempt to directly predict parasitic parameters or other factors contributing to pre/post-layout differences. For example, ParaGraph [18] uses GNNs to predict parasitic parameters from mixed-signal schematics; GNNs are utilized [14] to predict capacitance in SRAM-type circuits; and Mlpares [19] employs a random forest model [20] for interconnect resistance and capacitance prediction on analog circuits. However, due to the complexity of parasitic effects, these works are often confined to restricted circuit scales and types. The field still lacks effective parasitic parameter prediction for large-scale general-purpose circuits and subsequent PPA inference.

B. Graph Neural Network

Graph neural networks (GNNs) are models designed for graph-structured data. They learn representations by passing and aggregating information between nodes [21]–[23], with the message passing mechanism as their core [24], [25].

Synchronous and Asynchronous Message Passing. Synchronous message passing GNNs (SMP-GNNs) update all node representations simultaneously, constraining each node’s receptive field to the number of GNN layers. Conversely, asynchronous message passing GNNs (AMP-GNNs) iteratively update node subsets, allowing receptive fields to extend beyond layer counts and capture information from arbitrarily distant nodes until convergence [26].

III. PARAGATE

To enable a generalizable cross-stage timing and power prediction model, we propose ParaGate, a three-step coupled framework illustrated in Fig. 2. As shown in Fig. 2, the proposed method comprises three steps in the training phase. In

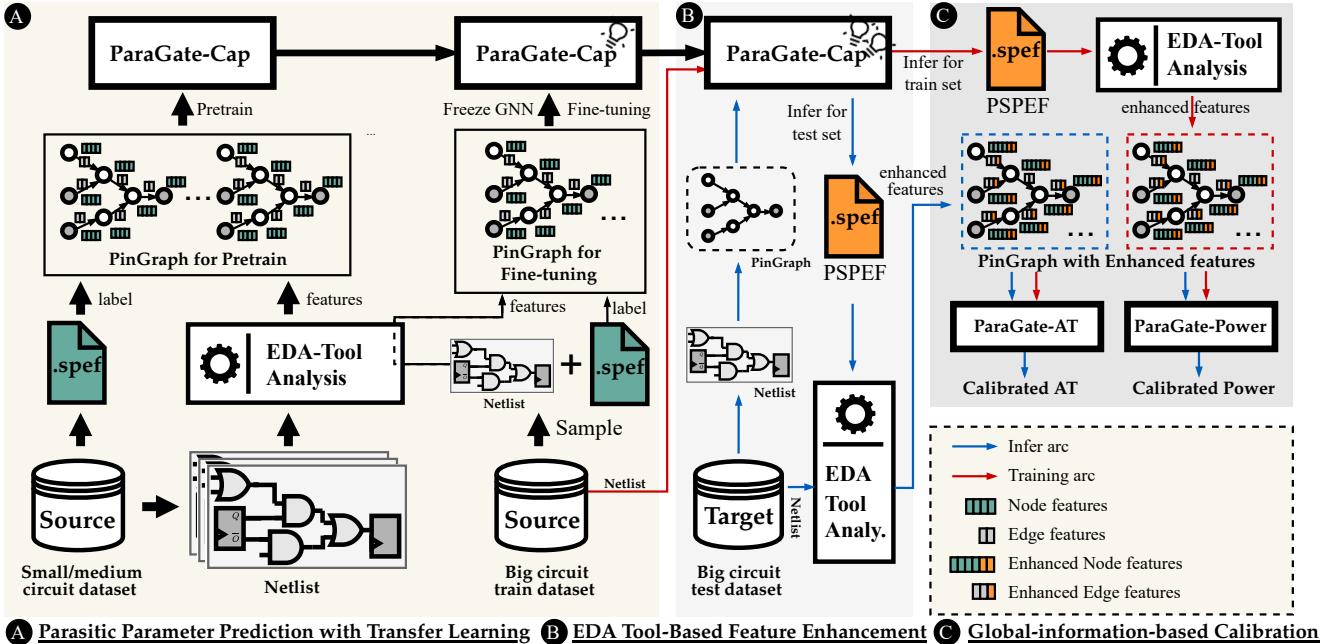


Fig. 2 ParaGate framework.

step 1, we incorporate circuit structural information to train our parasitic-parameter model, ParaGate-Cap, which involves two phases: model pretraining and subsequent fine-tuning. Next, in step 2, the EDA tool’s inference engine refines PPA metrics at the circuit level. Finally, in step 3, we leverage large-scale circuit training data along with the refined PPA-annotated netlist to adjust timing and power predictions using subgraph-level information. During inference, for a given input circuit, we first employ the fine-tuned ParaGate-Cap model to predict parasitic parameters. We then feed these predictions into the EDA tool, which propagates timing (arrival time, AT) and power estimates through the circuit topology. Finally, ParaGate-AT and ParaGate-Power are utilized to correct and refine these predictions. Key innovations of the proposed framework include:

- **Three-step decoupling for cross-stage PPA prediction.** (**Fig. 2 A - B - C**): By separating the workflow into three steps, each subproblem can be effectively addressed to improve model generalizability.
- **Parasitic parameter prediction with Transfer Learning** (**Fig. 2 A**): Our parasitic-modeling approach focuses on local interconnects and inherently learns the behavior of the routing tool. Since routing characteristics can vary significantly across different circuit specifications, our “pretraining + fine-tuning” strategy first learns universal patterns before adapting to extreme cases.
- **Global-information-based Calibration** (**Fig. 2 C**): Practical routing entails considerations such as regional density, so global circuit context is essential for accurate parameter predictions. To this end, after the initial EDA inference, we propose a subgraph-based model to correct arrival time and power results, further improving prediction precision.

A. Upstream - Parasitic Parameter Prediction with Transfer Learning

To more accurately represent the netlist, we construct a PinGraph to capture the circuit’s structural information, and then employ an SMP-GNN for model fitting. To better learn routing behaviors, we devise a pretraining + fine-tuning framework: the model first acquires generalizable post-routing patterns, and then, via fine-tuning, adapts to task-specific nuances—thereby achieving more precise modeling outcomes.

Graph Data Construction. To accurately capture AT and power propagation, it is critical to traverse the signal pathways within the circuit, which may have multi-fanout structures. Consequently, instead of using a cell-level graph, we adopt a PinGraph. Additionally, for the parasitic parameter model, since the netlist before layout lacks physical distances information, we opted for the lumped-capacitance modeling approach among the classic models (distributed model, the lumped-capacitance model, and a combined lumped capacitance-resistance model). Finally, as depicted in Fig. 3, we build a PinGraph based on the lumped-capacitance framework. By comprehensively embedding these logical and electrical characteristics of the gate-level netlist into its nodes and edges, the PinGraph enables the model to capture latent patterns during EDA’s place-and-route phase, thereby facilitating effective prediction of parasitic capacitance.

Model Architecture. The parasitic parameter prediction model consists primarily of an SMP-GNN, named ParaGate-Cap, and a readout layer. ParaGate-Cap’s aggregator design draws inspiration from graph attention networks (GAT) [27], specifically designed to learn the underlying model of interconnect parasitic parameters from the local topological connectivity of the gate-level netlist. The aggregation process is governed by

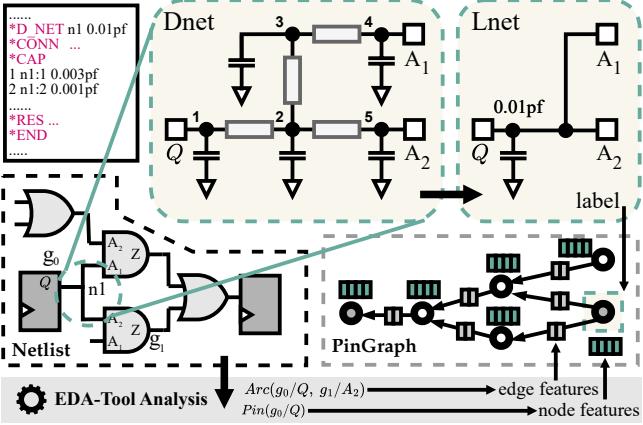


Fig. 3 Generation of PinGraph features and label.

the following equations:

$$\begin{aligned} \alpha_{ij} &= \text{softmax}_j((\mathbf{W}_q \mathbf{h}_i^{(k)})^\top \cdot (\mathbf{W}_k \mathbf{h}_j^{(k)} + \mathbf{W}_e \mathbf{e}_{ij})), \\ \mathbf{m}_j &= \mathbf{W}_v \mathbf{h}_j^{(k)} + \mathbf{W}_e \mathbf{e}_{ij}, \\ \mathbf{msg}_i &= \text{aggr}(\mathbf{h}_j^{(k)} \mid j \in \mathcal{N}(i)) = \sum_{j \in \mathcal{N}(i)} (\alpha_{ij} \cdot \mathbf{m}_j), \end{aligned} \quad (1)$$

where, $\mathcal{N}(i)$ denotes the neighbors of node i , α_{ij} signifies the attention weight between node i and node j , with \mathbf{e}_{ij} representing the edge attribute between them, and \mathbf{W}_q , \mathbf{W}_k , \mathbf{W}_v , \mathbf{W}_e are the weight matrices of the aggregator. The resulting aggregated message, \mathbf{msg}_i , is then fed into a gated recurrent unit (GRU) [28] to update the node's embedding.

$$\mathbf{h}_i^{(k+1)} \leftarrow \text{GRU}(\mathbf{h}_i^{(k)}, \text{aggr}(\mathbf{h}_j^{(k)} \mid j \in \mathcal{N}(i))). \quad (2)$$

These updated features are then passed through a multi-layer perceptron (MLP) [29] readout layer for final regression prediction of the parasitic parameters.

Pretraining and Fine-Tuning Framework. Effectively transferring a parasitic-parameter prediction model across circuits of varying scales poses a significant challenge, as models trained solely on abundant small/medium-scale circuits fail to learn the unique topological structures of large-scale designs. To address this issue, we propose a tailored transfer-learning strategy that combines specialized sampling with fine-tuning.

As shown in Fig. 2, we first pretrain on a dataset mainly composed of small and medium circuits, and subsequently fine-tune on large-scale circuits. In the pretraining phase, we construct features and labels for each circuit based on its netlist and standard parasitic exchange form (SPEF) data. Specifically, the features derive from the post-synthesis EDA report, while the labels capture post-place-and-route information. By training on a substantial volume of small/medium circuits, we optimize the graph regression pipeline—comprising aggregator, GRU, and MLP—yielding our pretrained ParaGate model.

During fine-tuning, we focus on large-scale circuits that contain extreme data points but have limited samples. To concentrate on high-discrepancy regions and avoid overfitting, we only sample and train on nodes with the most pronounced errors. We assume the primary adjustments for extreme val-

ues lie in the information-update mechanism and the readout MLP, while local neighbor aggregation patterns remain largely generalizable. Concretely, we decompose large circuits into sub-circuits, run inference with the pretrained model, and gather gradient information. We then select the top 20% of sub-circuits with the largest average gradients for subsequent fine-tuning. In this stage, we only train the MLP and GRU parameters while keeping the aggregator fixed, thus preserving the pretrained model's capacity to capture generalized local topological structures.

B. Midstream - EDA Tool-Based Feature Enhancement

PSPEF Construction and EDA Toolchain Integration. To integrate ParaGate with the EDA toolchain, we constructed ParaGate's pseudo standard parasitic exchange format (PSPEF) file based on the parasitic parameter prediction model, strictly adhering to the SPEF standard format. This file primarily uses the D_NET keyword to mark the predicted capacitance for each net while maintaining accurate connectivity.

PSPEF-Based Feature Enhancement. Utilizing the PSPEF file, the gate-level netlist is input into EDA tools to perform power analysis (PA) and static timing analysis (STA), thereby collecting enhanced features.

Leveraging PSPEF files and advanced EDA tool analysis reports, this step of ParaGate provides preliminary power and timing attributes for the gate-level netlist post-parasitic back-annotation. Crucially, it decouples the easily transferable parasitic parameters from the more challenging PPA attributes, furnishing the downstream task-specific calibration with physically meaningful and highly relevant enhanced features.

C. Downstream - Global-information-based Calibration

At this stage, the downstream model learns from the enhanced features derived from the netlist topology. These features originate from EDA tool analysis reports generated after back-annotating predicted parasitic parameters, and their overall trends align with ground truth. Consequently, the model effectively captures the discrepancies between preliminary EDA tool reports and actual values, enabling more precise calibration of PPA predictions.

Calibration Model Architecture. The calibration for ATs relies on ParaGate-AT, while cell-wise power calibration uses ParaGate-Power. Similar to ParaGate-Cap, both ParaGate-AT and ParaGate-Power employ an SMP-GNN. These models each comprise an aggregator built with GAT, an updater constructed with GRU, and a final MLP readout layer to regress the calibration targets.

$$\mathcal{L}_{AT/POWER} = \frac{1}{N_t} \sum \left(\hat{y} - \frac{y_{AT/Power}}{y_{raw_AT/raw_Power}} \right)^2 \quad (3)$$

Calibration Model Features. The calibration model's features are primarily composed of two sets. First, features reported by EDA tools from power and timing analysis on the netlist before parasitic back-annotation fully preserve the gate-level netlist's logical and electrical characteristics. Second, features from the

EDA tool analysis reports are obtained after back-annotating the PSPEF file output by the parasitic parameter prediction model from the first stage. The latter set, by incorporating the impact of parasitic parameters, reflects the shifts in netlist characteristics like power and timing. Combining these two feature types allows the calibration model to more comprehensively understand netlist behavior, enabling precise correction of PPA prediction results.

IV. EXPERIMENTAL RESULTS

We first detail our experimental setup in Section A, including training and test dataset configurations, model settings, and computational environment details. Subsequently, in Section B, we present the overall predictive performance of ParaGate along with comparisons to existing solutions. Finally, Sections C and D delve into ablation studies that analyze our staged prediction approach and the proposed fine-tuning framework.

A. Experimental Setup

This section details our experimental setup, covering dataset composition and generation, the code framework and model parameters, and the configurations for both comparative performance evaluations and ablation studies.

Dataset Construction. The dataset, as detailed in TABLE I, is partitioned into pretraining, training, and testing subsets. Specifically, the pretraining subset comprises 2004 small/medium-scale circuit instances, collected from public repositories such as GitHub and Hugging Face, with each instance ranging from 100 to 10,000 cells in size. The combined training and testing subsets consist of 14 open-source designs, with sizes ranging from 10,000 to 2,400,000 cells. To mitigate the impact of macro cells, units such as RAM and ROM were manually excluded from these designs.

Circuit Data Processing using EDA Tools. Data processing utilized the TSMC 28nm technology library. The flow began with design synthesis via Synopsys Design Compiler’s *compile_ultra* command. Place and route, performed using Cadence Innovus, then culminated in SPEF file generation. Subsequently, Synopsys PrimeTime generated final power and timing reports based on these SPEF files.

Code Framework and model configuration. Leveraging PyTorch Geometric, we developed a parser for efficient netlist/feature-to-tensor conversion for GNNs, and GNN output to PSPEF for EDA tools. ParaGate integrates three GNNs (shared hyperparameters, differing in input feature dimensions). PinGraph node features are initialized to a 128-D latent representation. GNN layers aggregate information (128-D Q, K, V, E vectors; softmax-weighted sums) and update node latent vectors via a 128-D GRU. After 8 shared-weight GNN layers, latent vectors feed into a 3-layer MLP (256-D hidden) for final regression. Experiments were run on 8 × NVIDIA A100 GPUs.

B. Overall Performance of ParaGate and Comparisons

Comparison Settings. We train ParaGate on the training sets listed in TABLE I and evaluate timing and power on the corresponding test sets. For benchmarking, we compare against

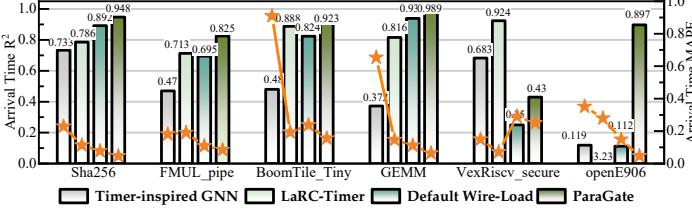
TABLE I Benchmark

Dataset	Circuit Name	Input Information		
		#node	#edge	#DFF
test	Sha256	19,346	72,180	1,112
	FMUL_pipe	60,815	236,749	2,395
	BoomTile_Tiny	157,022	515,210	19,726
	GEMM	293,672	1,088,965	20,031
	VexRiscv_secure	420,674	1,497,966	74,650
	openE906	2,380,540	6,932,091	310,630
train	VexRiscv_small	11,728	38,052	1,706
	RISCVmini	20,865	74,506	1,624
	SodorCore	24,983	94,250	1,668
	openE902	33,943	135,975	4,016
	FlexDPE	67,998	278,129	2,113
	FPU	135,725	573,539	1,543
	VexRiscv_full	502,154	1,509,160	76,232
	RocketTile_Tiny	973,877	2,870,165	168,555
pretrain	-	2,340,911	7,424,599	210,631

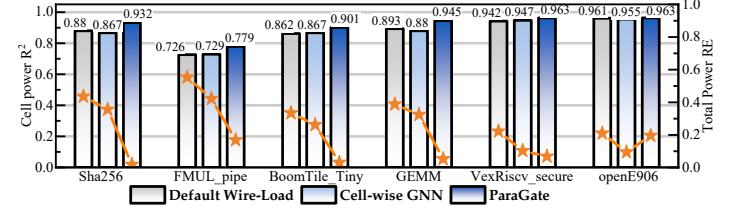
the latest methods: reproduced versions of LaRC-Timer [7] and Timer-Inspired GNN [8] for timing predictions, and the Cell-wise GNN model from [9] for power predictions, all with their physical parameter features removed. We also utilize an EDA tool with a wire-load model for timing and power predictions.

Performance Comparison. The comprehensive evaluation results are presented in Fig. 4, where Fig. 4(a) shows timing metrics and Fig. 4(b) shows power metrics. Overall, ParaGate outperforms baseline models on most benchmarks in both AT and Power prediction except *VexRiscv_secure*, demonstrating strong generalization and modeling accuracy. For AT prediction, ParaGate achieves higher R^2 and lower MAPE across nearly all designs, with a standout improvement on *openE906*, where it raises R^2 from 0.119 to 0.897. For Power prediction, ParaGate improves cell-wise R^2 by 2% to 5% and reduces total power relative error to below 10% for most designs. On *BoomTile_Tiny*, it achieves a relative error of only 0.909%. These results confirm ParaGate’s superior performance and robustness across a wide range of design scenarios. It is worth noting that in Fig. 4, we mainly compare against the existing cross-stage methods. For a more comprehensive comparison, we also implemented a parasitic parameters predictor [14] and paired it with our subsequent two steps, and compared in Section IV-C.

Inference-Time Analysis. We also report the runtime for each of the three ParaGate inference time in TABLE II. For comparison, we adopt the method proposed in [8] for timing prediction and the approach in [9] for power prediction as our baselines, measuring the prediction runtime and reporting the results in TABLE II. On average, ParaGate is about 1.14× slower than the baseline. This overhead arises because the baseline employs single-step end-to-end inference, whereas ParaGate utilizes a three-step pipeline to boost prediction generalization. Thanks to employing a synchronous GNN in ParaGate—whereas existing end-to-end prediction models must resort to a computationally inefficient asynchronous GNN to emulate timing analysis—the relative runtime overhead compared to existing end-to-end prediction is acceptable. For instance, on a 10k-cell circuit, the prediction time grows from 11.5s to 16.8s, and on the 2M-cell circuit *openE906*, it grows from 864.9s to 881.3s. Notably,



(a) DFF’s arrival time R^2 and MAPE comparison.



(b) Cell-wise power R^2 and total power relative error comparison.

Fig. 4 Overall Performance of ParaGate and Comparisons.

performing a full place-and-route flow to obtain performance metrics for this circuit can take tens of hours.

TABLE II Runtime (s) Comparison.

Circuit	Baseline	ParaGate	P&R
Sha256	11.5	16.8 (7.9+0.9+8.0)	1067.0
FMUL_pipe	67.9	85.6 (42.0+0.9+42.7)	4533.0
BoomTile_Tiny	61.7	68.9 (34.0+2.4+32.5)	3362.0
GEMM	133.8	226.1 (104.0+4.0+118.1)	7457.0
VexRiscv_secure	195.2	247.2 (122.0+6.8+118.4)	9501.0
openE906	864.9	881.3 (415.5+34.0+431.8)	47812.0

C. Ablation Study on Three-Step Framework

TABLE III Ablation Setting.

Variants	Pretraining	EDA-tool	Calibration
ParaGate w/o P	✗	✓	✓
ParaGate w/o E	✓	✗	✓
ParaGate w/o C	✓	✓	✗

Ablation Settings. As shown in TABLE III, “ParaGate w/o P” adopts the capacitance prediction model from [14] and is trained directly on the training set as detailed in TABLE I, without employing a pre-training and fine-tuning approach. “ParaGate w/o E” replaces the reasoning of EDA tools by training an AMP-GNN. “ParaGate w/o C” directly adopted the version of ParaGate without calibration.

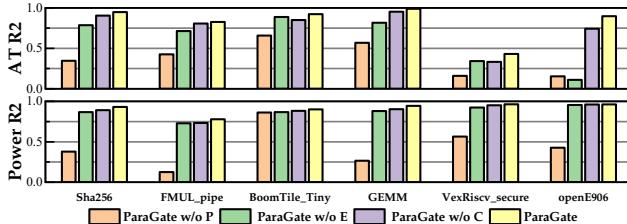


Fig. 5 ParaGate ablation study.

Effect of Pretraining. As shown in Fig. 5, “ParaGate w/o P” consistently underperforms notably due to overestimated capacitance, suggesting that training directly on large-scale circuits leads to overfitting and fails to capture realistic parasitic distributions. This highlights the necessity of pretraining on diverse small or medium scale designs.

Effect of EDA-Tool Reasoning. As depicted in Fig. 5, “ParaGate w/o E” demonstrates lower prediction accuracy across all samples compared to the full version. The GNN predicting enhanced features within “ParaGate w/o E” inherently contains errors. This imprecision may arise from the conflict between

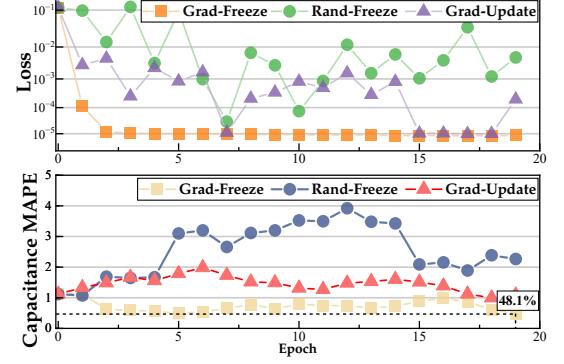


Fig. 6 Fine-tuning and sampling strategy ablation study.

neural networks’ statistical fitting and the exactness of structured computation.

Effect of Calibration. “ParaGate w/o C” shows lower accuracy than ParaGate. For example, on *openE906*, calibration improves AT R^2 from 0.742 to 0.897, with typical gains ranging from 1% to 7%.

D. Ablation Study on Fine-tuning and Sampling Strategy

Ablation Settings. We investigated the impact of sampling and fine-tuning on transfer learning performance using three methods based on our pre-trained parasitic parameter predictor: Grad-Freeze (20% max-average gradient samples, frozen aggregator), Ran-Freeze (20% random samples, frozen aggregator), and Grad-Update (20% max-average gradient samples, full parameter update).

Effect of Gradient-Based Sampling. Fig. 6 illustrates that Rand-Freeze exhibits a more oscillatory loss curve and converges to a poorer accuracy compared to Grad-Freeze. This observation suggests that indiscriminately selecting training samples is ineffective in enhancing model capability. Conversely, gradient-based sampling proves beneficial in identifying truly informative samples from the training set that are crucial for improving the model’s generalization ability.

Effect of Freezing Aggregator Parameters. Fig. 6 also demonstrates that Grad-Freeze achieves a faster and more stable convergence rate on the loss curve compared to Grad-Update. This indicates that maintaining the aggregator parameters fixed contributes to training stability. Furthermore, as shown in TABLE IV, Grad-Freeze yields superior prediction accuracy during testing compared to Grad-Update, signifying that fixed

aggregator parameters are conducive to enhancing the model's generalization capability.

TABLE IV Capacitance MAPE.

Circuit	Grad-Freeze	Rand-Freeze	Grad-Update
Sha256	68.7%	101.0%	93.5%
FMUL_pipe	43.4%	125.9%	123.5%
BoomTile_Tiny	32.7%	166.7%	142.9%
GEMM	108.8%	166.7%	124.6%
VexRiscv_secure	29.9%	176.5%	107.9%
openE906	39.1%	237.8%	117.4%
Average	44.3%	217.6%	116.7%

V. CONCLUSION

This paper addressed the critical challenge of AI model generalization for early PPA estimation in unseen, large-scale circuits. We proposed ParaGate, a novel transfer learning framework that effectively decouples PPA prediction by combining scale-agnostic parasitic parameter learning with precise EDA tool-based reasoning. Our experiments demonstrate ParaGate's superior generalization performance with minimal fine-tuning, marking a significant step towards robust PPA assessment in complex circuit designs.

REFERENCES

- [1] L. Chen, Y. Chen, Z. Chu, W. Fang, T.-Y. Ho, R. Huang, Y. Huang, S. Khan, M. Li, X. Li *et al.*, "The dawn of ai-native eda: Opportunities and challenges of large circuit models," *arXiv preprint arXiv:2403.07257*, 2024.
- [2] W. Fang, J. Wang, Y. Lu, S. Liu, Y. Wu, Y. Ma, and Z. Xie, "A survey of circuit foundation model: Foundation ai models for vlsi circuit design and eda," *arXiv preprint arXiv:2504.03711*, 2025.
- [3] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong *et al.*, "Machine learning for electronic design automation: A survey," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 5, pp. 1–46, 2021.
- [4] K. Chang, M. Chen, Y. Chen, Z. Chen, D. Fan, J. Gong, N. Guo, Y. Han, Q. Hao, S. Hou *et al.*, "Large processor chip model," *arXiv preprint arXiv:2506.02929*, 2025.
- [5] W. Fang, Y. Lu, S. Liu, Q. Zhang, C. Xu, L. W. Wills, H. Zhang, and Z. Xie, "Masterrtl: A pre-synthesis ppa estimation framework for any rtl design," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.
- [6] M. Wang, Y. Wen, B. Sun, J. Mu, J. Li, X. Wang, J. J. Ye, B. Yu, and H. Li, "Bridging layout and RTL: Knowledge distillation based timing prediction," in *Forty-second International Conference on Machine Learning*, 2025. [Online]. Available: <https://openreview.net/forum?id=pW925fKyK>
- [7] X. He, Z. Fu, Y. Wang, C. Liu, and Y. Guo, "Accurate timing prediction at placement stage with look-ahead rc network," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1213–1218.
- [8] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin, "A timing engine inspired graph neural network model for pre-routing slack prediction," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1207–1212.
- [9] Y. Du, Z. Guo, X. Jiang, Z. Chai, Y. Zhao, Y. Lin, R. Wang, and R. Huang, "Powpredict: Cross-stage power prediction with circuit-transformation-aware learning," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.
- [10] R. Zhong, J. Ye, Z. Tang, S. Kai, M. Yuan, J. Hao, and J. Yan, "Preroutgnn for timing prediction with order preserving partition: Global circuit pre-training, local delay learning and attentional cell modeling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 15, 2024, pp. 17087–17095.
- [11] Z. Wang, S. Liu, Y. Pu, S. Chen, T.-Y. Ho, and B. Yu, "Restructure-tolerant timing prediction via multimodal fusion," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.
- [12] Z. Lyu, X. Li, Z. Du, and Q. Guo, "Explainable and layout-aware timing prediction," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–9.
- [13] Z. Chai, Y. Zhao, W. Liu, Y. Lin, R. Wang, and R. Huang, "Circuitnet: An open-source dataset for machine learning in vlsi cad applications with improved domain-specific evaluation metric and learning strategies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 12, pp. 5034–5047, 2023.
- [14] S. Shen, D. Yang, Y. Xie, C. Pei, B. Yu, and W. Yu, "Deep-learning-based pre-layout parasitic capacitance prediction on sram designs," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, 2024, pp. 440–445.
- [15] W. R. Davis, P. Franzon, L. Francisco, B. Huggins, and R. Jain, "Fast and accurate ppa modeling with transfer learning," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–8.
- [16] Y. Zhang, X. Chen, Y. Yang, A. Ramamurthy, B. Li, Y. Qi, and L. Song, "Can graph neural networks help logic reasoning?" *arXiv preprint arXiv:1906.02111*, 2019.
- [17] ———, "Efficient probabilistic logic reasoning with graph neural networks," *arXiv preprint arXiv:2001.11850*, 2020.
- [18] H. Ren, G. F. Kokai, W. J. Turner, and T.-S. Ku, "Paragraph: Layout parasitics and device parameter prediction using graph neural networks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.
- [19] B. Shook, P. Bhansali, C. Kashyap, C. Amin, and S. Joshi, "Mlparest: Machine learning based parasitic estimation for custom circuit design," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

- [20] G. Biau and E. Scornet, “A random forest guided tour,” *Test*, vol. 25, no. 2, pp. 197–227, 2016.
- [21] Z. Liu and J. Zhou, *Introduction to graph neural networks*. Springer Nature, 2022.
- [22] B. Khemani, S. Patil, K. Kotecha, and S. Tanwar, “A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions,” *Journal of Big Data*, vol. 11, no. 1, p. 18, 2024.
- [23] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks,” *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.
- [24] Z. Zheng, Y. Bei, S. Zhou, Y. Ma, M. Gu, H. Xu, C. Lai, J. Chen, and J. Bu, “Revisiting the message passing in heterophilous graph neural networks,” *arXiv preprint arXiv:2405.17768*, 2024.
- [25] H. He, X. Yu, J. Zhang, S. Song, and K. B. Letaief, “Message passing meets graph neural networks: A new paradigm for massive mimo systems,” *IEEE Transactions on Wireless Communications*, vol. 23, no. 5, pp. 4709–4723, 2023.
- [26] L. Faber and R. Wattenhofer, “Asynchronous neural networks for learning in graphs,” *arXiv preprint arXiv:2205.12245*, 2022.
- [27] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.
- [28] R. Dey and F. M. Salem, “Gate-variants of gated recurrent unit (gru) neural networks,” in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE, 2017, pp. 1597–1600.
- [29] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, “Multilayer perceptron and neural networks,” *WSEAS Transactions on Circuits and Systems*, vol. 8, no. 7, pp. 579–588, 2009.