

# ForgeEDA: A Comprehensive Multimodal Dataset for Advancing EDA

Zhengyuan Shi<sup>1,9</sup>, Zeju Li<sup>1,9</sup>, Chengyu Ma<sup>2,9</sup>, Yunhao Zhou<sup>1,9</sup>, Ziyang Zheng<sup>1,9</sup>, Jiawei Liu<sup>3,9</sup>, Hongyang Pan<sup>4,9</sup>, Lingfeng Zhou<sup>5,9</sup>, Kezhi Li<sup>1,9</sup>, Jiaying Zhu<sup>1,9</sup>, Lingwei Yan<sup>6,9</sup>, Zhiqiang He<sup>6,9</sup>, Chenhao Xue<sup>7,9</sup>, Wentao Jiang<sup>2,9</sup>, Fan Yang<sup>4</sup>, Guangyu Sun<sup>7</sup>, Xiaoyan Yang<sup>5</sup>, Gang Chen<sup>6</sup>, Chuan Shi<sup>3</sup>, Zhufei Chu<sup>2</sup>, Jun Yang<sup>8,9</sup> and Qiang Xu<sup>1,9\*</sup>

<sup>1</sup>*Department of Computer Science and Engineering, The Chinese University of Hong Kong, Sha Tin, Hong Kong S.A.R.*

<sup>2</sup>*Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo, China*

<sup>3</sup>*School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China*

<sup>4</sup>*School of Microelectronics, State Key Laboratory of Integrated Chips and System, Fudan University, Shanghai, China*

<sup>5</sup>*School of Electronic and Information Engineering, Hangzhou Dianzi University, Hangzhou, China*

<sup>6</sup>*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China*

<sup>7</sup>*School of Integrated Circuits, Peking University, Beijing, China*

<sup>8</sup>*School of Integrated Circuits, Southeast University, Nanjing, China*

<sup>9</sup>*National Center of Technology Innovation for EDA, Nanjing, China*

Corresponding Author: Qiang Xu (qxu@cse.cuhk.edu.hk)

**Abstract**—We introduce ForgeEDA, an open-source comprehensive circuit dataset across various categories. ForgeEDA includes diverse circuit representations such as Register Transfer Level (RTL) code, Post-mapping (PM) netlists, And-Inverter Graphs (AIGs), and placed netlists, enabling comprehensive analysis and development. We demonstrate ForgeEDA’s utility by benchmarking state-of-the-art EDA algorithms on critical tasks such as Power, Performance, and Area (PPA) optimization, highlighting its ability to expose performance gaps and drive advancements. Additionally, ForgeEDA’s scale and diversity facilitate the training of AI models for EDA tasks, demonstrating its potential to improve model performance and generalization. By addressing limitations in existing datasets, ForgeEDA aims to catalyze breakthroughs in modern IC design and support the next generation of innovations in EDA.

**Index Terms**—Open-source, multimodal benchmarks, logic synthesis, AI for EDA.

## I. INTRODUCTION

Electronic Design Automation (EDA) is an indispensable technology in the semiconductor industry, breaking down the complexity of integrated circuit (IC) design into a multi-stage process, including logic synthesis, technology mapping, placement, routing, and layout generation. Over the past several decades, researchers in the EDA field have worked to address the challenges in reducing design costs, enhancing chip PPA metrics, and ensuring design functionality. Thanks to the advanced EDA solutions [1] and the emerging AI for EDA (AI4EDA) solutions [2], [3], engineers are now capable

of designing large-scale and diverse ICs, from billion-transistor processors and AI accelerators to compact low-power chips.

However, despite these achievements, the hardware design ecosystem remains highly closed compared to software community. The lack of accessible data has been a significant barrier to evaluating novel EDA technologies and training AI4EDA models, which in turn has hindered the rapid progress of the entire hardware design. Existing datasets [4]–[9] offer open-access circuits but suffer from limitations in size and variety. For example, the largest circuit in ITC99 benchmark [5] contains only ten thousands of gates and CircuitNet [8], [9] only includes processors, making them insufficient for addressing the needs of modern IC design. Additionally, datasets like [10] collect web-scale hardware code snippets for training large language models (LLMs). While valuable for specific applications such as RTL understanding and generation, they do not support the remaining EDA stages and are thus limited to narrow use cases. These shortcomings highlight a critical gap in existing EDA datasets, underscoring the need for a comprehensive, practical, and large-scale dataset to support the next generation of EDA advancements.

To address the above issues, we propose ForgeEDA, a comprehensive multimodal dataset designed to advance EDA research. ForgeEDA spans various circuit formats across the EDA flow and encompasses a wide range of real-world chip types. Specifically, the dataset includes 6 categories, 20 divisions, and 1,189 repositories that reflect real-world circuit designs, such as RISC-V cores, AI accelerators, arithmetic units, encoders/decoders, interface modules, and control units (see Table. I). In addition to the original Verilog code, ForgeEDA provides the corresponding Post-Mapping (PM) netlists, placed netlists, and And-Inverter Graphs (AIGs), generated using EDA tools.

This work was partly supported by the General Research Fund of the Hong Kong Research Grants Council (RGC) (Grant Nos. 14212422 and 14202824), National Technology Innovation Center for EDA, National Natural Science Foundation of China (Grant Nos. 62032001, U20B2045, U1936220, 62192784, 62172052, 62002029, 61772082, 92373207), Beijing Natural Science Foundation (Grant No. L243001), and the 111 Project (Grant No. B18001).

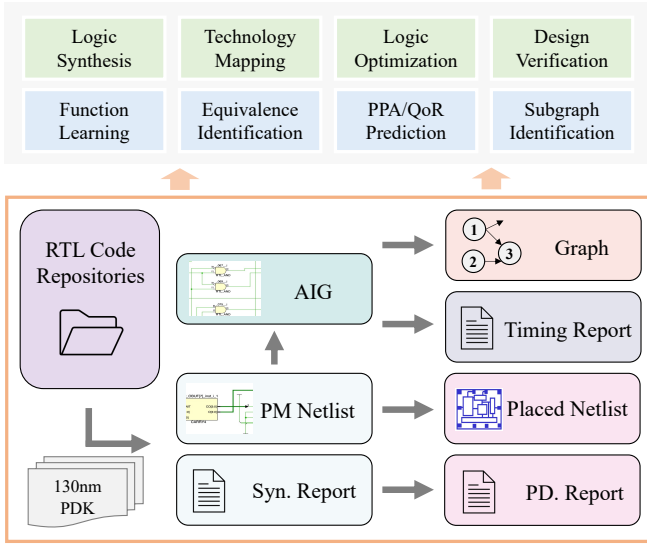


Fig. 1. Overview of ForgeEDA

TABLE I  
CATEGORIES OF COLLECTED DESIGNS

Category	Division	Counts	Category	Division	Counts
Processor	RISCV	111	Interface	SPI	45
	MIPS	26		UART	40
	AI	53		PCI	30
	Others	58		Others	32
arithmetic	Adder	38	Controller	DMA	30
	Multipiler	109		PWM	94
	Shifter	15		Arbiter	43
	Counter	66		Clock Control	31
	Others	66		Others	35
Encoder/Decoder		182	Others		85
<b>Total</b>		<b>1,189</b>			

We utilize ForgeEDA to assess EDA solutions, focusing on logic synthesize and optimization. Our evaluation reveals a significant performance gap between open-source EDA solutions and advanced commercial tools, a disparity that is less evident in previous datasets due to their narrow scopes or scales. Additionally, we leverage our dataset for training AI4EDA solutions. The expanded scale and diversity of training data enable our dataset to enhance existing AI4EDA models, driving further improvements in their performance. We believe that our dataset can pave the way for advancing the next generation of EDA.

## II. RELATED WORK

### A. Previous Dataset in EDA

A comprehensive and unified circuit dataset is essential for the fair evaluation of EDA solutions and serve as an infrastructure for training AI4EDA models. Some existing datasets focus on specific circuit formats tailored for a single task, like the EPFL benchmark [6] for logic synthesis or a dataset comprising collected RTL code snippets [10] utilized

to fine-tune large language models (LLMs) for Verilog code generation and comprehension. Additionally, prominent organizations host contests or annual competitions and release corresponding benchmarks such as ISCAS89 [4], ITC99 [5], HWMCC [11], and SAT competitions [12]. However, these datasets are limited in the types and scales of circuits and are often constrained to single-task evaluations. Other datasets provide multiple circuit representations. For instance, OpenABC-D [7] presents a comprehensive circuit dataset for logic synthesis, encompassing RTL code, circuit netlists, and graph-based representations, whereas the CircuitNet [8], [9] extends the EDA process to the physical design phase by including floorplans and layouts. Nevertheless, these datasets have constraints in their coverage. OpenABC-D, for example, is limited to only 29 open-source hardware IP designs, while CircuitNet primarily focuses on processors. Different from the previous datasets, our ForgeEDA covers 7 categories and 22 sub-categories of circuit designs, providing RTL code, netlists and AIGs.

### B. EDA Applications

1) *Logic Synthesis*: Logic synthesis is a cornerstone of EDA design flow, enabling the transformation of high-level hardware description languages (HDL) into optimized netlists of logic gates [13]. Traditional logic synthesis focuses on balancing design constraints and meeting design objectives such as minimizing area, power consumption, and delay [14], [15]. Over time, advances in this field have introduced techniques for fine-tuning synthesis processes to better align with downstream physical design stages, improving efficiency throughout the design flow [16], [17]. However, the de-facto pipeline for evaluating logic synthesis methodologies is heavily reliant on narrow and small-scale benchmarks [4], [6]. Performance on these benchmarks has nearly converged, making it difficult to assess the generalization capability of synthesis approaches across more diverse and complex scenarios. We leverage our ForgeEDA dataset to evaluate a range of logic synthesis methods, providing a more rigorous and diverse testing ground that mirrors real-world design challenges.

2) *PPA Prediction*: PPA prediction has emerged as a critical focus in early-stage IC design due to its potential to streamline design cycles by enabling informed decision-making. Recent research has focused on developing pre-synthesis PPA estimation frameworks. For example, MasterRTL [18] introduces a bit-level design representation, the Simple Operator Graph (SOG), which is readily available from RTL code and enables more precise PPA predictions. Another notable framework, SNS [19], leverages a path-based approach by employing deep learning models to estimate PPA at the path level. The existing approaches are typically evaluated on a limited range of circuit types and focus on PPA metrics at post-synthesis stages, lacking the capacity to account for the full physical design process. The physical design stage is both time-intensive and critical for accurate PPA prediction, as it reflects a more realistic measure of the final design's performance and constraints. Currently, no

benchmark provides an integrated dataset encompassing RTL code, netlists, and corresponding physical design reports.

3) *AI for EDA*: The integration of AI into EDA has opened new avenues for automating and optimizing various design processes [2], [3]. With the advanced capabilities of AI models, AI4EDA solutions have demonstrated remarkable achievements in circuit optimization [20]–[22], design metric prediction [23], [24] and hardware code generation [25]. Besides, an emerging area within AI4EDA is circuit representation learning [26]–[29], which learns general representations of circuit designs that can be easily transferred across various EDA tasks. Moreover, DeepGate3 [30] proves the data scaling ability of circuit representation learning models, indicating that the model performance can be further improved with more training data. Clearly, a critical prerequisite for the success of AI4EDA solutions is access to high-quality, diverse datasets that encompass a wide range of circuit types and design stages. In this work, we train a series of models with ForgeEDA to demonstrate its utility and effectiveness.

### III. DATASET

#### A. Overview

The overview of our proposed dataset is illustrated in Fig 1. To construct the dataset, we first collect 1,189 RTL code repositories from the Internet. The categories of the designs in our dataset are summarized in Table I. Next, we employ EDA tools to generate various circuit formats, including post-mapping (PM) netlists and And-Inverter Graphs (AIGs), accompanied by logs detailing PPA metrics. All the contents of our dataset are summarized in Table II. Finally, we evaluate our dataset in both practical EDA applications and AI for EDA tasks.

#### B. Data Preparation

1) *Data Collection*: We begin by identifying and extracting design-related keywords from chip specifications available on AllDatasheet [31], a website hosting millions of semiconductor datasheets. This approach ensures that our dataset covers a diverse array of design types, reflecting real-world chip applications. Next, we gather repositories in Verilog (.v) from the Internet based on these selected keywords to build the RTL code dataset [32], [33]. Finally, we filter out repositories that cannot be synthesized or contain only post-mapping Verilog files, refining our dataset to include only those with comprehensive and synthesizable designs. We totally collect 1,189 high-quality open-source repositories in Verilog and categorize them into 6 fields.

2) *Logic Synthesis and Physical Design*: We employ Synopsys Design Compiler and Skywater 130nm [34] Process Design Kit (PDK) to synthesize RTL code repositories into post-mapped (PM) netlists, along with detailed synthesis reports. To expand our dataset, each module is treated as the top module in its respective synthesis flow. As a result, a total of 4,450 netlists are generated. Following synthesis, we utilize Cadence Innovus to carry out the floorplanning and placement steps in the ASIC physical design flow. This generates a placed netlist and a physical design report. After placement, the

precise timing information can be extracted since the distances between ports and cells are established.

3) *Graph Generation*: We begin by converting PM netlists, based on the standard cell library in the PDK, into And-Inverter Graphs (AIGs) using the ABC tool [35]. Next, we apply the Static Timing Analysis (STA) tool, *stime*, within ABC to generate the timing report. To support the AI4EDA solution, we construct a dataset using graphs represented in PyTorch Geometric [36]. These graphs are derived from the PM netlists and AIGs. We also provide the sub-AIGs for model training, which are randomly extracted sub-circuits with 500-5,000 nodes. Totally, 83,155 sub-circuits and their graph representations are generated for model training.

### IV. PRACTICAL EDA APPLICATIONS

To validate the effectiveness of our dataset, we perform a comprehensive evaluation to measure the performance gap between open-source and commercial synthesis tools. For this assessment, we employed a variety of advanced open-source tools and commercial EDA solutions across the following three main logic synthesis tasks.

#### A. RTL Synthesis

We performed a complete synthesis flow starting from RTL input, which included logic optimization and technology mapping, to generate gate-level netlists. We then evaluated and compared the area and delay metrics of these netlists. Specifically, we analyzed the performance of the commercial tool Design Compiler's `compile_ultra` command (DCU) [37] and the open-source logic synthesis tool Yosys [38]. Table III summarizes the post-synthesis delay and area metrics for both methods, which is reported by STA command *stime* in ABC [35]. Out of 20 benchmarks, Yosys failed to produce results for 4 cases due to limitations in its verilog parser, a common drawback in many academic tools. For the 16 benchmarks that Yosys successfully processed, the tool achieved an average delay of  $1.20\times$  and an area of  $1.77\times$  compared to DCU. These results highlight the significant scalability advantage of commercial tools like DCU over academic tools, while also suggesting that academic tools, such as Yosys, tend to perform better in delay optimization than in area optimization.

#### B. AIG Optimization

To evaluate the capabilities of academic tools in AIG synthesis, we focused on both AIG optimization and AIG technology mapping. We use the selected circuits in Table III in the following experiments. AIG optimization, which aims to convert an initial AIG into a more optimized form, is assessed with area and delay metrics. We approximate area reduction using the number of nodes reduced and delay reduction using the logic level reduced.

To comprehensively assess the performance of these methods, we compared the following four approaches:

- `resyn2rs/compress2rs`: Script focuses on minimizing AIG nodes without increasing the logic depth.

TABLE II  
CONTENTS OF FORGEEDA DATASET

Modality	#Samples	Format	Contents
Code Repository	1,189	.v	Verilog code with annotated comments and specifications
PM Netlist	4,450	.v .rpt	Post-mapping netlist Reports of area, path delay and power
Placed Netlist	4,450	.v / .def .rpt	Placed netlist Reports of area, WNS, TNS and power
AIG	4,450	.aig	Raw AIG files
Sub-AIG	83,155	.aig .npz	Raw AIG files Parsed graphs in PyTorch Geometric

TABLE III  
RTL SYNTHESIS RESULTS

Circuit		DCU		Yosys	
		area ( $\mu m^2$ )	delay(ps)	area ( $\mu m^2$ )	delay(ps)
Processor	riscv1	189,973.45	10,596.26	195,852.83	36,679.18
	riscv2	122,628.86	63,897.15	764,719.69	7,772.28
	riscv3	20,111.79	17,155.09	41,422.23	10,554.09
	imageProc	552,800.19	16,575.54	1,765,114.12	182,101.42
Arithmetic	alu1	3,160.53	6,197.87	-	-
	alu2	1,631.56	10,570.08	41,684.98	140,038.53
	alu3	36,488.75	138,830.16	43,481.70	130,999.24
	alu4	17,027.58	14,461.29	17,516.80	6,059.55
En/Decoder	qcLdpc	3,589,661.50	42,341.13	-	-
	ldpc	368,947.05	8,386.78	235,082.95	6,011.34
	viterbi	147,456.42	82,473.52	-	-
	dct	49,444.92	14,142.23	72,768.54	18,663.73
Interface	pcie1	118,623.77	10,276.88	140,099.38	8,975.54
	pcie2	53,663.97	9,011.58	-	-
	pcie3	61,327.57	105,812.45	94,431.82	33,475.29
	axis	4,569.38	12,909.13	3,687.29	4,893.00
Controller	memctrl	181,391.47	10,191.19	432,793.81	2,774.81
	branch	41,039.36	3,927.65	51,365.51	129,410.59
	signalTab	15,531.15	2,838.90	18,611.60	3,370.16
	arbiter	3,168.04	1,291.89	4,332.91	1,435.12
Geomean		48,523.05	14,604.15	71,987.59	15,944.40
Imp.		<b>1.00</b>	<b>1.00</b>	<b>1.77</b>	<b>1.20</b>

- *if -g*: Logic refactoring algorithm that reduces network depth by utilizing sum-of-product (SOP) expressions [39].
- *orchestrate*: Combines multiple operations into a single traversal of the AIG for enhanced coordination and optimization [40].

In Fig. 2, each node corresponds to an AIG optimized by a specific approach, with its coordinates representing area reduction and delay reduction. The origin of the coordinate system represents the original, unoptimized AIG. The geometric mean of optimization performance across all benchmarks is depicted as ‘\*’. The results reveal several key observations. First, *resyn2rs* and *compress2rs* demonstrate the most balanced optimization performance, achieving significant improvements in both area and delay across benchmarks. Second, *if -g* excels in delay reduction, achieving an average of 31% logic level reduction, outperforming all other methods in this metric. Third, *orchestrate*, despite being a recent method,

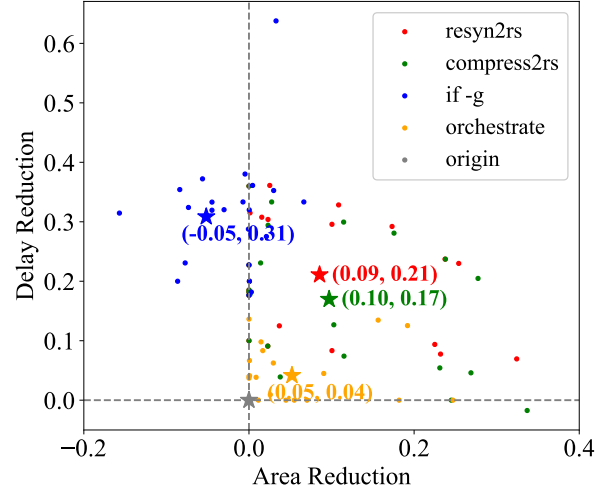


Fig. 2. AIG Optimization Results

performs the worst, indicating limitations in its generalization ability for the various benchmarks.

### C. AIG Technology Mapping

We used the optimized AIG as input to generate mapped netlists and evaluated the mapping efficiency based on netlist area and delay. As illustrated in Fig. 3, the baseline approach is *abc* (map).

To assess mapping performance, we compared the ABC tool with DCU, using the ABC map command as a baseline. Additionally, we evaluated SOTA mapping commands such as *&nf*, *choice* [41] within ABC. The results indicate that, despite achieving similar performance in a few benchmarks, ABC shows a significant performance gap compared to commercial tools like DCU in most cases. On average, the mapped netlists produced by ABC are approximately 10% worse in terms of area and delay, highlighting the challenges academic tools face in closing the gap with commercial solutions.

## V. AI FOR EDA TASKS

The application of deep learning techniques in EDA has emerged as an attractive research direction, garnering significant interest from both the AI and EDA communities [2], [3]. Our ForgeEDA serves a dual purpose: as a dataset for training AI4EDA models and as a benchmark for evaluating their performance. We perform a series of representative tasks to train

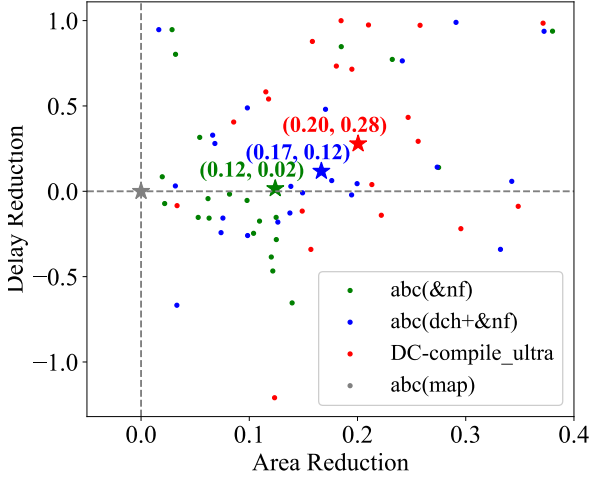


Fig. 3. AIG Technology Mapping Results

TABLE IV  
COMPARISON OF LOSS VALUE FOR PROBABILITY PREDICTION ACROSS DIFFERENT DATASET SPLITS

Model	Full Dataset	10% Dataset	1% Dataset
GCN [44]	0.0683	0.1661	0.2497
GAT [45]	0.1579	0.3470	0.3775
GraphSAGE [46]	0.0307	0.0596	0.2199
DeepGate2 [27]	0.0335	0.0466	0.1965
PolarGate [42]	0.0148	0.0164	0.0192

existing circuit learning models and assess their effectiveness, including probability prediction [27], [42] and equivalent gate identification [27], [42], [43] in our experiments.

#### A. Probability Prediction

1) *Task Statement*: Probability Prediction involves determining the functionality of circuits by predicting the logic-1 probability of each gate under random simulation. This task plays a crucial role in analyzing circuit reliability and testability under probabilistic scenarios. As such, it is regarded as one of the most representative pre-training tasks in circuit learning models [27], [42].

2) *Experiment Setting*: Given an AIG graph  $G$ , we encode  $G$  using Graph Neural Network (GNN) models as follows:

$$[h_1, h_2, \dots, h_n] = GNN(G), \quad (1)$$

where  $h_i$  represents the embedding of gate  $i$ .

Next, for a specific gate  $k$ , we predict its logic-1 probability using a 3-layer Multi-Layer Perceptron (MLP):

$$L_{gate}^{prob} = MAE(p_k, MLP_{prob}(h_k)), \quad (2)$$

where MAE denotes the mean absolute error. The task performance is evaluated across various GNN models: GCN [44], GAT [45], GraphSAGE [46], DeepGate2 [27], and PolarGate [42].

We further divide our training dataset into 10% and 1% subsets to evaluate the data scalability of the models and assess the effectiveness of our expanded dataset.

TABLE V  
COMPARISON OF LOSS VALUE FOR EQUIVALENT GATE IDENTIFICATION TASK ACROSS DIFFERENT DATASET SPLITS

Model	Full Dataset	10% Dataset	1% Dataset
GCN [44]	0.4285	0.4396	0.4641
GAT [45]	0.4266	0.4358	0.4729
GraphSAGE [46]	0.2789	0.4400	0.5815
DeepGate2 [27]	0.0813	0.1528	0.2781
PolarGate [42]	0.0759	0.1032	0.1320

3) *Result Analysis*: Table IV summarizes the performance results, where lower MAE indicates better performance. First, we observe that PolarGate achieves the best performance with an MAE of 0.0148, significantly outperforming other models. Second, as the dataset size increases, the MAE of all models decreases further. For instance, the GraphSAGE model achieves an MAE of 0.2199 on the 1% dataset, 0.0596 on the 10% dataset, and 0.0307 on the full dataset. These results demonstrate that ForgeEDA provides substantial benefits to AI4EDA solutions by offering a large-scale training dataset.

#### B. Equivalent Gate Identification

1) *Task Statement*: Equivalent gate identification focuses on determining the functional similarity between two gates based on their truth tables [27]. This task is pivotal for applications such as Logic Equivalence Checking (LEC), SAT solving, and optimization of logic synthesis workflows.

2) *Experiment Setting*: Given an AIG graph  $G$ , we encode  $G$  using GNN models:

$$[h_1, h_2, \dots, h_n] = GNN(G), \quad (3)$$

where  $h_i$  represents the embedding of gate  $i$ .

For a given pair of gates  $(i, j)$ , their functional similarity is predicted using cosine similarity between their embeddings:

$$L_{gate}^{tt\_pair} = MAE(D_{(i,j)}^{gate\_tt}, \text{cossim}(h_i, h_j)), \quad (4)$$

where  $D_{(i,j)}^{gate\_tt}$  denotes the ground-truth functional similarity derived from the truth table, and  $\text{cossim}(h_i, h_j)$  computes the cosine similarity. The task performance is evaluated using MAE across models: GCN [44], GAT [45], GraphSAGE [46], DeepGate2 [27], and PolarGate [42]. It should be noted that we do not train and evaluate FGNN [43], as it predicts a hard label (equivalent or not) rather than the similarity between two logic gates.

3) *Result Analysis*: Table V presents the performance comparison (lower MAE is better). The results demonstrate that domain-specific models, such as DeepGate2 [27] and PolarGate [42], significantly outperform general-purpose GNNs. For instance, PolarGate achieves an MAE of 0.0759, compared to the much higher MAE of 0.4285 observed for GCN. Similarly, increasing the scale of the training data leads to further reductions in MAE across all models, highlighting the importance of dataset size.

## VI. CONCLUSION

In this work, we present ForgeEDA, a comprehensive multimodal dataset designed to advance research and development in EDA. The dataset comprises 6 categories, 20 divisions, and 1,189 code repositories of real-world circuit designs. ForgeEDA includes multiple circuit formats, such as post-mapping netlists generated by logic synthesis tools, placed netlists produced by physical design tools, and graph representations tailored for open-source EDA tools and circuit learning models. Using ForgeEDA, we evaluate existing EDA solutions in logic synthesis and optimization, identifying significant opportunities for further advancements. Additionally, training AI4EDA models with our dataset demonstrates improved performance due to the large scale and diversity of training samples provided. We hope this work accelerates the transition of EDA towards open-source development and supports the creation of next-generation solutions for the community.

## REFERENCES

- [1] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.
- [2] G. Huang, J. Hu, Y. He, J. Liu *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.
- [3] L. Chen, Y. Chen, Z. Chu *et al.*, “Large circuit models: opportunities and challenges,” *Science China Information Sciences*, vol. 67, no. 10, pp. 1–42, 2024.
- [4] F. Brglez, D. Bryan, and K. Kozminski, “Notes on the iscas’89 benchmark circuits,” *North-Carolina State University*, 1989.
- [5] S. Davidson, “Characteristics of the itc’99 benchmark circuits,” in *ITSW*, 1999.
- [6] L. Amarú, P.-E. Gaillardon, and G. De Micheli, “The epfl combinational benchmark suite,” in *IWLS*, no. CONF, 2015.
- [7] A. B. Chowdhury, B. Tan, R. Karri, and S. Garg, “Openabc-d: A large-scale dataset for machine learning guided integrated circuit synthesis,” *arXiv preprint arXiv:2110.11292*, 2021.
- [8] Z. Chai, Y. Zhao, Y. Lin, W. Liu, R. Wang, and R. Huang, “Circuitnet: An open-source dataset for machine learning applications in electronic design automation (eda),” *arXiv preprint arXiv:2208.01040*, 2022.
- [9] X. Jiang, Y. Zhao, Y. Lin, R. Wang, R. Huang *et al.*, “Circuitnet 2.0: An advanced dataset for promoting machine learning innovations in realistic chip design environment,” in *ICLR*, 2024.
- [10] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, “Benchmarking large language models for automated verilog rtl code generation,” in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.
- [11] A. Biere, N. Froleyks, and M. Preiner, “Hardware model checking competition 2024,” 2024.
- [12] M. Iser and C. Jabs, “Global Benchmark Database.” [Online]. Available: <https://benchmark-database.de/>
- [13] S. Hassoun and T. Sasao, *Logic synthesis and verification*. Springer Science & Business Media, 2001, vol. 654.
- [14] A. Mishchenko, R. Brayton *et al.*, “Scalable don’t-care-based logic optimization and resynthesis,” *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 4, no. 4, pp. 1–23, 2011.
- [15] A. Mishchenko *et al.*, “Dag-aware aig rewriting a fresh look at combinational logic synthesis,” in *Design Automation Conference*, 2006, pp. 532–535.
- [16] L. Zhu and X. Guo, “Delay-driven physically-aware logic synthesis with informed search,” in *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE, 2023, pp. 327–335.
- [17] H. Pan, C. Lan, Y. Liu, Z. Wang, L. Shang, X. Zeng *et al.*, “Physically aware synthesis revisited: guiding technology mapping with primitive logic gate placement,” *arXiv preprint arXiv:2408.07886*, 2024.
- [18] 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.
- [19] C. Xu, C. Kjellqvist, and L. W. Wills, “Sns’s not a synthesizer: a deep-learning-based synthesis predictor,” in *International Symposium on Computer Architecture*, 2022, pp. 847–859.
- [20] A. B. Chowdhury, M. Romanelli, B. Tan, R. Karri, and S. Garg, “Retrieval-guided reinforcement learning for boolean circuit minimization,” in *International Conference on Learning Representations*.
- [21] J. Song, A. Swope *et al.*, “Circuitvae: Efficient and scalable latent circuit optimization,” *arXiv preprint arXiv:2406.09535*, 2024.
- [22] Z. Shi, M. Li, S. Khan, L. Wang, N. Wang, Y. Huang, and Q. Xu, “Deeptpi: Test point insertion with deep reinforcement learning,” in *2022 IEEE International Test Conference (ITC)*. IEEE, 2022, pp. 194–203.
- [23] D. Liu and B. C. Schafer, “Efficient and reliable high-level synthesis design space explorer for fpgas,” in *International Conference on Field Programmable Logic and Applications*. IEEE, 2016, pp. 1–8.
- [24] E. Ustun, C. Deng, D. Pal, Z. Li, and Z. Zhang, “Accurate operation delay prediction for fpga hls using graph neural networks,” in *international conference on computer-aided design*, 2020.
- [25] S. Liu, W. Fang, Y. Lu *et al.*, “Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution,” in *2024 IEEE LLM Aided Design Workshop (LAD)*, 2024, pp. 1–5.
- [26] M. Li, S. Khan, Z. Shi *et al.*, “Deepgate: Learning neural representations of logic gates,” in *Design Automation Conference*, 2022, pp. 667–672.
- [27] Z. Shi, H. Pan, S. Khan, M. Li, Y. Liu, J. Huang *et al.*, “Deepgate2: Functionality-aware circuit representation learning,” in *2023 IEEE/ACM International Conference on Computer Aided Design*. IEEE, 2023.
- [28] S. Khan, Z. Shi, M. Li, and Q. Xu, “Deepseq: Deep sequential circuit learning,” in *DATE*. IEEE, 2024, pp. 1–2.
- [29] Z. Zheng, S. Huang, J. Zhong, Z. Shi, G. Dai, N. Xu, and Q. Xu, “Deepgate4: Efficient and effective representation learning for circuit design at scale,” *arXiv preprint arXiv:2502.01681*, 2025.
- [30] Z. Shi, Z. Zheng, S. Khan, J. Zhong, M. Li, and Q. Xu, “Deepgate3: Towards scalable circuit representation learning,” in *2024 IEEE/ACM International Conference on Computer Aided Design*. IEEE, 2024.
- [31] Alldatasheet, 2003. [Online]. Available: <https://www.alldatasheet.com/>
- [32] Z. Li, C. Xu, Z. Shi, Z. Peng, Y. Liu, Y. Zhou *et al.*, “Deepcircuitx: A comprehensive repository-level dataset for rtl code understanding, generation, and ppa analysis,” *arXiv preprint arXiv:2502.18297*, 2025.
- [33] Y. Liu, C. Xu, Y. Zhou, Z. Li, and Q. Xu, “Deeptl: Bridging verilog understanding and generation with a unified representation model,” *arXiv preprint arXiv:2502.15832*, 2025.
- [34] T. Edwards, “Introduction to the skywater pdk: The new age of open source silicon.” [Online]. Available: <https://skywater-pdk.readthedocs.io/>
- [35] A. Mishchenko *et al.*, “Abc: A system for sequential synthesis and verification.”
- [36] M. Fey and J. E. Lenssen, “Fast graph representation learning with pytorch geometric,” *arXiv preprint arXiv:1903.02428*, 2019.
- [37] Synopsys, “Design compiler ultra,” <https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html>, 2024.
- [38] C. Wolf, “Yosys open synthesis suite,” <https://github.com/YosysHQ/yosys>, 2016.
- [39] A. Mishchenko, R. Brayton, S. Jang, and V. Kravets, “Delay optimization using sop balancing,” in *International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2011, pp. 375–382.
- [40] Y. Li *et al.*, “Dag-aware synthesis orchestration,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.
- [41] A. Mishchenko, R. Brayton, and S. Jang, “Global delay optimization using structural choices,” in *FPGA’10*, 2010, pp. 181–184.
- [42] J. Liu, J. Zhai, M. Zhao, Z. Lin, B. Yu, and C. Shi, “Polargate: Breaking the functionality representation bottleneck of and-inverter graph neural network,” in *International Conference on Computer Aided Design*, 2024.
- [43] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, “Functionality matters in netlist representation learning,” in *Design Automation Conference*, 2022, pp. 61–66.
- [44] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.
- [45] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.
- [46] W. Hamilton *et al.*, “Inductive representation learning on large graphs,” *Advances in neural information processing systems*, vol. 30, 2017.