

Graph-SRAM: Efficient Graph Learning-based SRAM Simulation via Waveform Propagation

Beisi Lu, Lihao Liu, Li Shang *Member, IEEE* and Fan Yang* *Member, IEEE*

Abstract—High-speed SRAM arrays are essential for data-intensive Systems-on-Chip (SoCs). However, accurate timing characterization of these SRAMs requires transistor-level SPICE simulations, which are extremely time-consuming due to the large dimensions and complexity of modern designs. In this work, we present Graph-SRAM, an efficient graph learning-based simulation method using waveform propagation. Our approach models cells and interconnects as heterogeneous graphs and embeds global switching features, enabling a customized Graph Neural Network (GNN) to capture structural and functional patterns in both combinational and sequential circuits. Compared to HSPICE, Graph-SRAM achieves a significant speedup of $6905.32\times$ while maintaining high accuracy, with an average error of only 4.28% in predicting path waveforms.

Index Terms—graph neural network, SRAM, SPICE simulation, waveform prediction.

I. INTRODUCTION

SRAM plays a crucial role in modern digital integrated circuits and SoC designs [1]. Renowned for its high access speeds and compatibility with standard logic processes, SRAM serves as the ideal choice for on-chip caches and register files. As VLSI technology and AI applications advance, these high-performance SRAM arrays are occupying an increasingly significant portion of chip area in microprocessors and GPUs [2], [3]. Due to its wide range of applications, customized SRAM configurations are frequently adopted to meet stringent performance constraints, and their timing performance becomes critical for ensuring the overall functionality and reliability of circuit designs [4]–[6].

Transistor-level SPICE simulations are essential for accurate timing characterization of SRAMs. However, due to the extremely large scales of modern SRAM designs, simulating such circuits presents several major challenges:

- **High computational cost.** The runtime of SPICE simulations for SRAM circuits increases proportionally to $O(n^3)$, where n denotes the number of bitcells in the array [7], [8]. For customized SRAM designs, the simulation process may take several months or even be impossible, accounting for the majority of the design time.

This research is supported by National Natural Science Foundation of China (NSFC) research project 92373207.

Beisi Lu, Lihao Liu and Fan Yang are with State Key Lab of Integrated Chips and Systems, and College of Integrated Circuits and Micro-Nano Electronics, Fudan University, Shanghai, China. Li Shang is with State Key Lab of Integrated Chips and Systems, and School of Computer Science, Fudan University, Shanghai, China.

*Corresponding author: yangfan@fudan.edu.cn

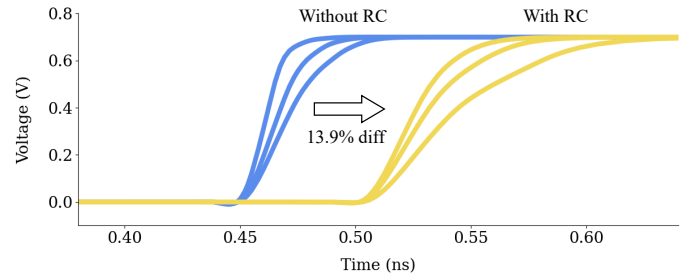


Fig. 1. The impact of interconnects parasitics on SRAM timing performances.

- **Significant parasitic effects.** Interconnects in large-scale SRAMs, especially at advanced technology nodes, can introduce substantial parasitic effects [8], [9] and cause significant timing degradation, as shown in Fig. 1. Long wordlines and bitlines contribute considerable resistance and capacitance (RC) to the read/write paths, further slowing down the simulation and complicating accurate analysis.

Traditional methods for accelerating the simulation process can generally be classified into three categories. The first is the critical path extraction-based approach [10], [11], where only the critical path is simulated instead of the whole circuit. A second category is the fast-SPICE-based method [12], which employs techniques such as model simplification and matrix partitioning to reduce simulation time. The third approach involves RC reduction, simplifying the RC networks of interconnects to lower computational complexity [13]. However, these methods still rely on time-consuming SPICE simulations, and the simplifications or reductions result in a loss of accuracy.

Recently, Machine Learning (ML) techniques have been widely adopted in the analysis of SRAMs and in the timing analysis of integrated circuits. For example, Bayesian Optimization (BO) has been employed for SRAM netlist reduction and yield optimization [14], [15]. Bouhlila et al. utilize Extreme Gradient Boosting Regressor (XGBR) to analyze SRAM stability [16], marking the first instance of its application in this context. Cao et al. employ a Transformer-based framework that effectively captures long-range delay correlations along a timing path [17], while Guo et al. develop a graph learning-based method that overcomes the limited receptive field of conventional graph models for slack prediction [18]. However, meeting industrial requirements for SRAMs demands accurate path waveform prediction, which provides richer information like slew rates and signal integrity. The methods described

above are not applicable to this type of waveform-level SRAM analysis.

To tackle the challenges mentioned above, we propose Graph-SRAM, an efficient graph learning-based SRAM simulation framework that delivers SPICE-level accuracy. Our approach is motivated by the observation that SRAMs consist of a large number of duplicated structures, such as bitcells, whose transistor-level architectures, together with interconnects, can naturally be represented as graphs. We model local cells and interconnects as heterogeneous graphs with edge features. While conventional graph modeling methods struggle to incorporate sequential elements such as bitcells, Graph-SRAM overcomes this limitation by embedding global features related to timing switching behavior into the graph. Graph Neural Networks (GNNs) are well-suited for learning local topological patterns in graphs. However, standard GNN architectures are insufficient for modeling such graphs with rich information in SRAMs. To this end, we propose a customized graph learning-based model with an attention mechanism that enables the model to learn varying relationships between these circuit components. Furthermore, Graph-SRAM employs a local-to-global learning approach, enabling it to capture local circuit behaviors and generalize this knowledge to full-path prediction by waveform propagation across the entire SRAM. The main contributions of this work are summarized as follows:

- We propose an effective heterogeneous graph modeling approach for SRAM designs, representing both cells and interconnects as graphs with edge features. By embedding global features related to timing switching behavior, our method is applicable to both combinational and sequential logic circuits.
- We develop a graph learning-based simulation framework, Graph-SRAM, to predict path-level waveforms in SRAMs. The customized GNN architecture, enhanced with an attention mechanism, effectively captures local structural patterns of cells and interconnects. Leveraging waveform propagation, our approach bridges local learning with global prediction, enabling accurate path-level waveform simulation of SRAMs.
- Experimental results show that, the proposed method achieves a significant speedup of **6905.32** \times over HSPICE, while maintaining high accuracy with an average path waveform prediction error of only **4.28**%. Furthermore, compared to a commercial fast-SPICE tool, our method achieves nearly the same accuracy while being **523.54** \times faster. The model is trained using only 30% of the total data and demonstrates strong generalization capability.

The remainder of this paper is organized as follows. Section II reviews the fundamental of GNNs. Section III details the proposed framework of Graph-SRAM. The experimental results are presented and analyzed in Section IV. Finally, Section V concludes the paper.

II. BACKGROUND

This section provides background on the fundamentals of GNNs.

GNNs have emerged as a foundational paradigm for processing graph-structured data and have proven to be powerful tools in EDA [19], [20]. Traditional GNNs learn graph embeddings through message passing and information updating. The Graph Attention Network (GAT) [21] introduces an attention mechanism that differentiates the relative importance of neighboring nodes, thereby enhancing the representational capacity of graph embeddings. The attention coefficient α_{ij} between node v_i and node v_j is computed as follows:

$$\alpha_{ij} = \text{softmax}_j(a(Wh_{v_i}, Wh_{v_j})), \quad (1)$$

where h_{v_i} and h_{v_j} are the embeddings of nodes v_i and v_j , respectively; W is a learnable weight matrix; a denotes the shared attention mechanism such as a feedforward neural network; and softmax_j indicates that the softmax operation is applied over all neighboring nodes v_j of node v_i . With this attention mechanism, the node update function at layer $l + 1$ is given by:

$$h_{v_i}^{l+1} = \phi \left(\sum_{v_j \in N_i} \alpha_{ij} W h_{v_j}^l \right), \quad (2)$$

where $h_{v_i}^l$ is the embedding of node v_i at layer l , and ϕ represents the node update function, such as a Multi-Layer Perceptron (MLP). Here, sum is used as the aggregation function.

III. PROPOSED GRAPH-SRAM METHOD

To accurately predict the output waveform of a cell following an input transition, in addition to the input waveform, two other essential inputs are needed: the cell's underlying circuit structure, and its logical output values immediately before and after the transition, which is critical for sequential elements like SRAM bitcells. This section introduces our proposed method as follows. We first briefly introduce the graph modeling of circuit structures. Then, we detail the unique challenges of waveform prediction in SRAM and our approach to obtain the logic transition information. The model architecture is then presented. Finally, we introduce the waveform propagation and training methodology.

A. Graph Representation for Standard Cells and Interconnects

Integrated circuits naturally exhibit a graph-like structure. Based on the circuit netlist, a graph $G = (V, E)$ can be built, where $V = \{v_1, v_2, \dots, v_n\}$ is the set of nodes and $E = \{e_{ij} | v_i, v_j \in V\}$ is the set of edges. Different construction schemes are designed for the standard cells in SRAM and the interconnects between them, as illustrated in Fig. 2. Graphs of standard cells are constructed by creating nodes for transistors and electrical nets. Transistor nodes are characterized by their physical properties (e.g., channel width, channel length and number of fins), while net nodes are characterized by their type (input, output, VDD, GND, and general net) and degree.

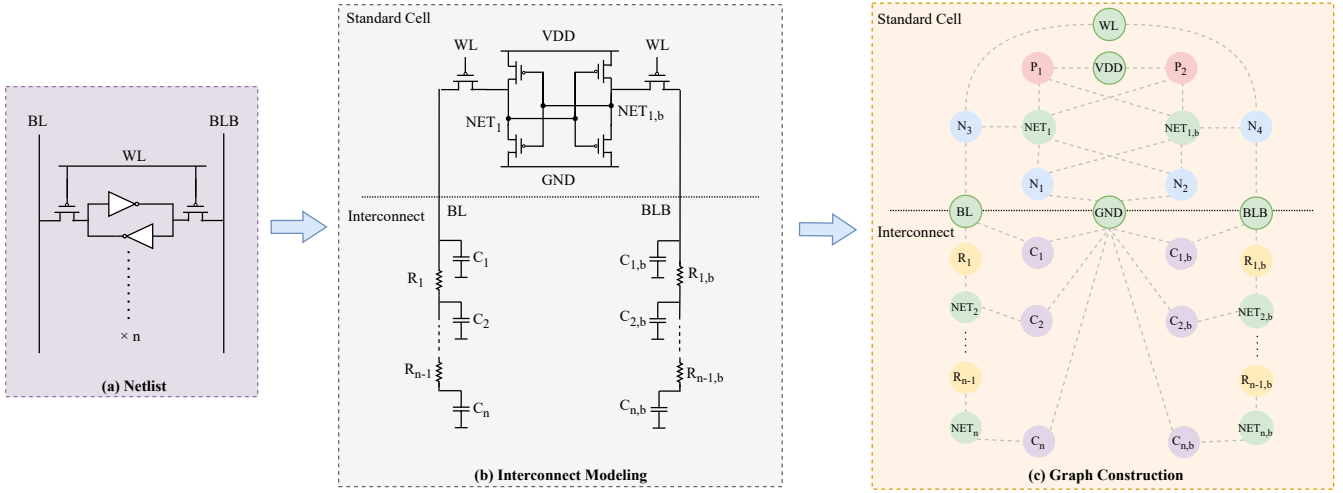


Fig. 2. Graph construction method for cells and interconnects. (a) SPICE-level netlist of a bitcell. (b) Interconnects are modeled using a lumped π -model. (c) Graph construction of standard cell and interconnect.

Without layout information, we use a lumped- π model to estimate the RC parasitics on the interconnect line. A node connecting N cells is modeled as a lumped- π model containing N capacitors and $N-1$ resistors, as shown in Figure 2(b). Resistors and capacitors are then represented as two types of nodes in graph with their corresponding values as features. While an approximation model is used here to estimate the parasitic effect, the proposed method is directly applicable to the post-layout netlist after parameter extraction without any adjustment.

After defining nodes, edges in the graph are constructed by physical connections between nodes and are one-hot encoded by connection type (e.g., net-to-drain, net-to-source, net-to-resistor).

Once the graphs for standard cells and interconnects are built, they are merged into a unified graph based on their physical connectivity, and the merged graph serves as the input for training, prediction, and waveform propagation. Our detailed graph representation method provides the model with a comprehensive understanding of the circuit's physical topology and parasitic effects.

B. Switch-Level Simulation for Logic Transition Information

A key challenge in SRAM waveform prediction is that a cell's output waveform is determined not only by its current input but also by its historical input. A typical SRAM read path, for instance, involves a signal propagating from a wordline driver, activating a bitcell's access transistor, and causing a voltage change on the bitline. The voltage change is then amplified and buffered to the output. When a read operation is executed, the output pattern of a bitcell depends on its currently stored bit value, that is, the value of the input data when the last write operation was performed on this bitcell.

To address this, a switch-level simulation can be performed. The simulation takes the SRAM's operational inputs and the logic functions of the standard cells, and calculates the logical

output values of each cell before and after every read or write operation, denoted as *data_before* and *data_new*. For a combinational element like an inverter, these values are obtained by directly inverting its input's logical transition. For a sequential element like a bitcell, the algorithm searches its historical input to determine these two values. Since only 0/1 logical calculations are involved, the simulation time is negligible. The simulation results are then used to specify the output transition type in the following waveform prediction, such as a rising or falling edge.

In addition, sink capacitance has a critical impact on the output waveform and delay of a standard cell. We approximate this impact using the cell's fanout, which is defined as the number of gates driven by the cell, and can be easily extracted from the netlist. The fanout is combined with *data_before* and *data_new* to form the global feature, serving as a crucial input to our prediction model. By incorporating the cell's logic transition information, the model is equipped to handle memory elements, a critical capability for performing accurate waveform prediction in SRAM analysis.

C. Graph-SRAM Model Architecture

Based on the constructed graph and our goal to predict waveforms rather than only delay values, we design and implement a graph learning model called Graph-SRAM, the overall architecture of which is shown in Fig. 3. The input comprises three components: the input waveform, the circuit graph $G = (V, E)$, and the global feature vector. Three feature extraction modules are designed to process these inputs:

1) *Waveform Feature Extractor*: To capture the dynamic characteristics of the input waveform, a Convolutional Neural Network (CNN)-based extractor is employed. Multi-channel convolution is adopted here to learn various patterns in the waveform and identify key features. The output of the CNN is flattened and passed through a fully connected layer to produce the waveform embedding.

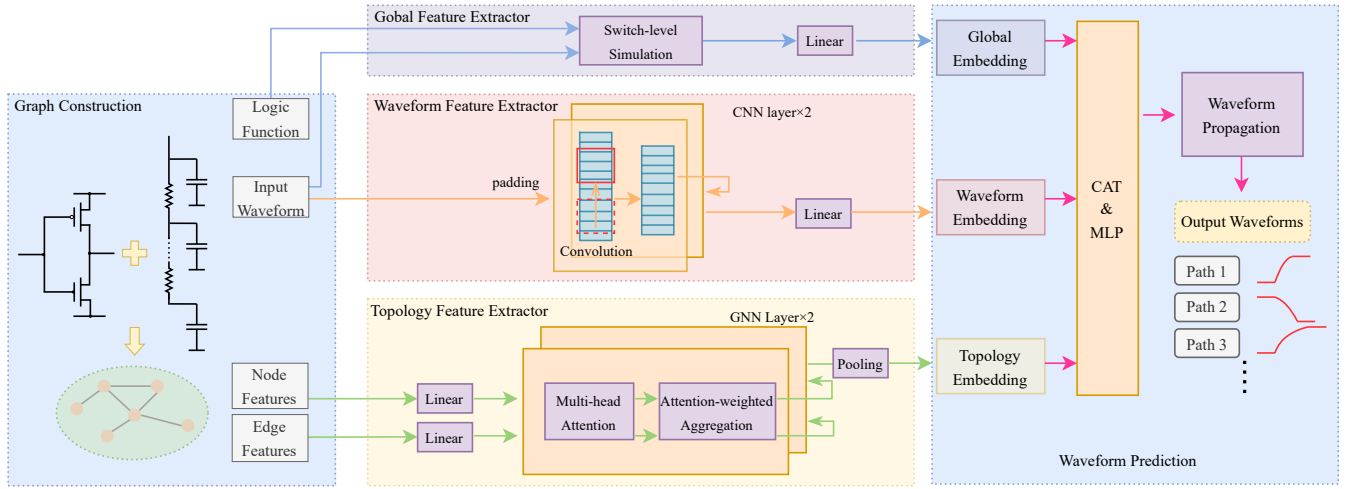


Fig. 3. The architecture of the proposed Graph-SRAM model.

2) *Topological Feature Extractor*: In this module, the initial node and edge features of the graph are projected into a high-dimensional space through linear transformation. These embedded features are then passed into a two-layer GNN with an attention mechanism. The GNN layers dynamically learn the importance of neighboring nodes based on their connectivity and assign varying attention weights, effectively aggregating neighborhood information and capturing interactions between devices. To further enhance its ability, edge features are considered when calculating the attention score, as follows:

$$\alpha_{ij} = \text{softmax}_j(a(Wh_{v_i}, Wh_{v_j}, W_e h_{e_{ij}})) \quad (3)$$

where W_e is a learnable matrix and $h_{e_{ij}}$ is the feature of the edge between nodes v_i and v_j .

3) *Global Feature Extractor*: A simple linear transformation encodes the global features (*data_before*, *data_new*, and fanout). This module is intentionally kept simple to preserve the direct and significant impact of these features on the output waveform.

The embedding vectors from these modules are concatenated and passed through a final MLP to predict the output waveform. This customized architecture allows the model to holistically process topological, temporal, and logic state-related information for a precise prediction.

D. Waveform Propagation and Training

When predicting voltage waveforms for an entire path, we employ a waveform propagation strategy, as shown in Fig. 4. We build a graph for each standard cell along the path with its associated interconnect. The predicted output waveform of one stage is fed as the input for the subsequent stage, continuing iteratively until the end of the path, with all paths in the design being processed in parallel. Each waveform is represented using 11 time points, which correspond to the times at which the voltage rises or falls to 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,

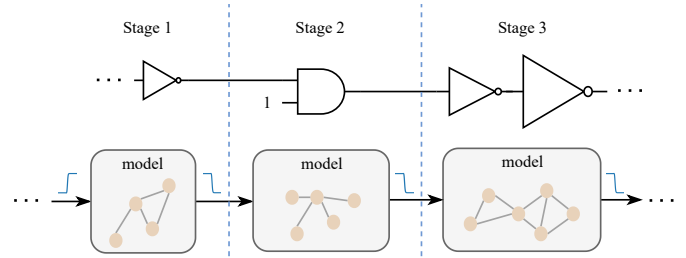


Fig. 4. Waveform propagation for path timing prediction.

0.8, 0.9, and 0.95 of VDD. This scheme allows for a unified treatment of both rising and falling waveforms.

For training, we use the Adam optimizer with a learning rate of 0.001 and a batch size of 4096, determined by the capability of the GPU. The model is trained to minimize the mean squared error (MSE) between its predictions and the golden labels generated by HSPICE simulations. The training process takes 561.65 seconds.

IV. EXPERIMENTS

A. Experiment Setup

To evaluate the performance of our proposed method across various SRAM sizes, we generated a dataset of 150 circuit netlists incorporating SRAM arrays of varying sizes. The numbers of rows and columns of the SRAMs were randomly sampled from the ranges of 128 to 256 and 32 to 64, respectively. Buffers with different MOSFET parameters (e.g., channel width, number of fins) were applied in different SRAM circuits to allow the model to learn the effects of transistor parameter changes. The designs were implemented using the ASAP 7nm FinFET PDK [22]. For each netlist, RC load networks were established for the interconnects as previously described.

An 8-cycle transient simulation was performed in HSPICE for each SRAM circuit, involving access to two distinct

TABLE I
RESULTS OF GRAPH-SRAM, FAST-SPICE AND HSPICE SIMULATIONS ON TEST NETLISTS

Method	Cell Error				Path Error						Time(s)	Normalized Time
	Waveform		Delay		Waveform		Read Delay		Write Delay			
	MAE (ps)	MARE (%)	MAE (ps)	MARE (%)	MAE (ps)	MARE (%)	MAE (ps)	MARE (%)	MAE (ps)	MARE (%)		
Graph-SRAM	1.50	3.09	0.93	1.71	1.79	4.28	5.33	1.90	1.82	2.06	10.71	1
Fast-SPICE	-				0.67	3.5	5.10	2.81	1.08	1.18	5607.10	523.54
HSPICE	-				-						73955.94	6905.32

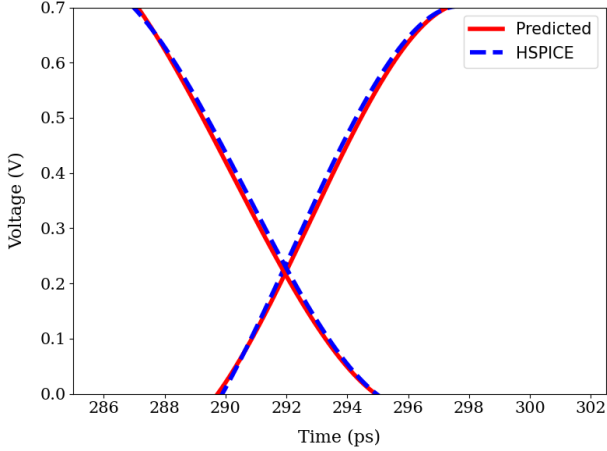


Fig. 5. 2 Predicted waveforms with 4.21% and 3.79% errors respectively compared with HSPICE.

addresses. For each address, a four-cycle “write-read-write-read” sequence was executed, with complementary data used for the two write operations. This stimulus was designed to sufficiently cover critical operational transitions within the SRAM circuit, thus ensuring the representativeness and diversity of the collected waveform dataset.

After simulations, the input and output waveforms for each standard cell and its corresponding interconnect were extracted from the results. Graph representations were also constructed following the method detailed in Section III-A. A training set was constructed using 30% of the waveform data, while the remaining 70% was allocated to the test set to evaluate the generalization capability of our model. Within the training set, 10% of the cells were randomly held out as a validation set. The accuracy of the model was quantified using mean absolute error (MAE) and mean absolute relative error (MARE), defined as:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (4)$$

$$\text{MARE} = \frac{1}{n} \sum_{i=1}^n \frac{|y_i - \hat{y}_i|}{|y_i|} \quad (5)$$

where n is the total number of predictions, y_i is the ground-truth label from HSPICE, and \hat{y}_i is the predicted value.

The experiments were conducted on a Linux server equipped with an Nvidia GeForce RTX 4090 GPU and an Intel Xeon Gold 6248R CPU operating at 2.6 GHz. Our model was implemented with the PyTorch framework, leveraging the PyTorch Geometric toolkit for graph-based operations.

B. Evaluation of Model Prediction Results

1) *Standard Cell Characterization*: We first evaluated the model’s accuracy in predicting the output waveforms of individual standard cells, where the input waveform for each cell was sourced directly from the HSPICE simulations. The delay of each cell was also extracted from the waveform and compared, which is defined as the time interval between the 50% transition points of the input and output signals.

The results are summarized in the *Cell Error* columns of TABLE I, in the form of MAE and MARE between the predicted values and the HSPICE results. Our model achieves high accuracy, with an overall MARE of only 3.09% for waveform and 1.71% for delay.

2) *Waveform Propagation Results*: Next, we assessed the model’s performance to predict the signals along entire timing paths. Here, the predicted output waveform of a cell serves as the input for the subsequent cell in the path, emulating real-world signal propagation. For delay analysis, we focused on the critical read-path and write-path delays, which were separately extracted and evaluated.

As shown in the *Path Error* columns of TABLE I, the results of our Graph-SRAM exhibit high similarity to the HSPICE ground truth. Across all netlists, the MARE for the propagated waveforms, read delay, and write delay is 4.28%, 1.90%, and 2.06%, respectively. Additionally, to benchmark our framework against fast-SPICE methodologies, we also present the results of CustomSim, a commercial fast-SPICE simulator, for comparison. Compared to CustomSim, Graph-SRAM shows a lower MARE for read delay, which is 1.90%. For the other metrics, its error is at most 0.88% higher, while providing a significant speedup. These results confirm that our model can make accurate timing predictions even with propagated inputs, maintaining high precision throughout complex circuit paths.

3) *Speed Analysis*: TABLE I also presents the inference time for full-path waveform propagation of the model versus fast-SPICE and HSPICE simulation time. Our model achieves a speedup of $523.54\times$ over fast-SPICE and $6905.32\times$ over

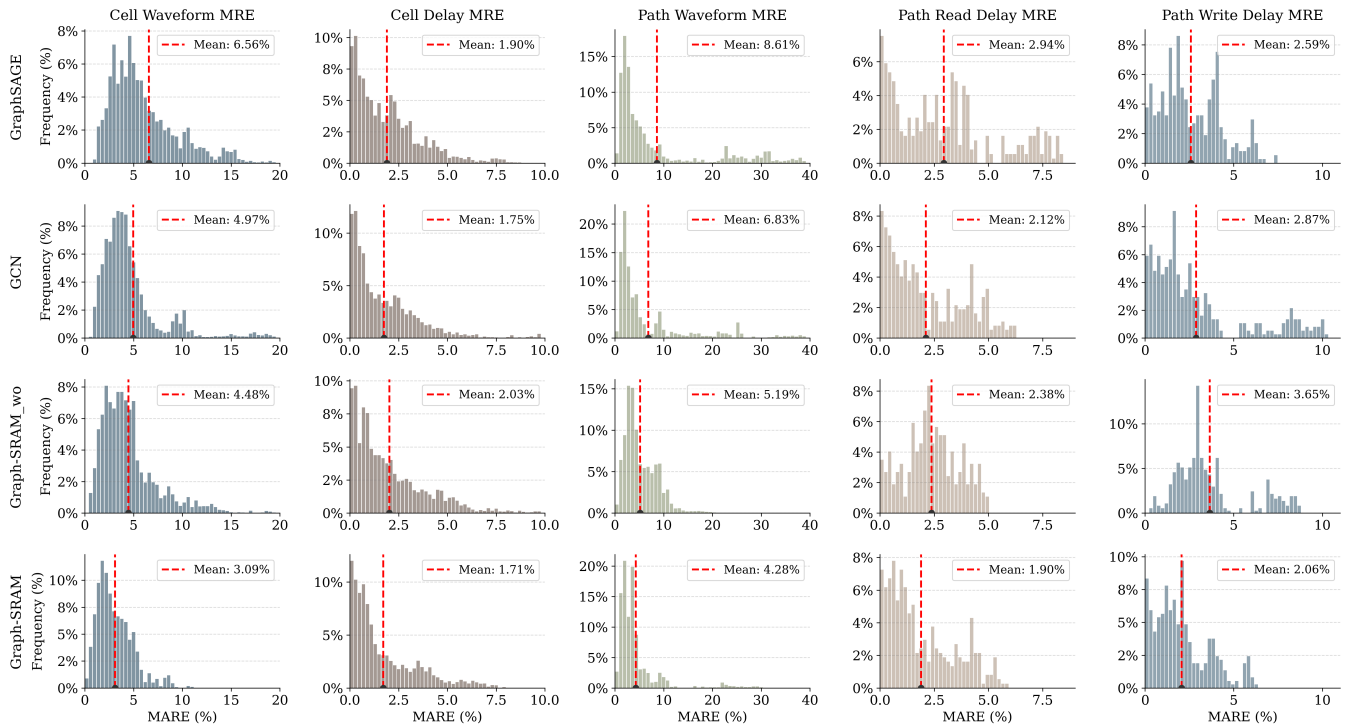


Fig. 6. Comparison with other graph-based ML method.

HSPICE, demonstrating its high computational efficiency. In addition, we separately tested the simulation speed on the smallest 128×32 and largest 256×64 SRAMs in the dataset. Our method takes 0.08s and 0.13s on the two netlists, while HSPICE takes 231.76s and 932.02s respectively. The calculated speedups are $2897\times$ and $7169\times$, respectively. This performance variation stems from the fundamental difference between the two methods. HSPICE relies on iterative numerical methods to solve large-scale systems of nonlinear differential equations. In contrast, the inference time of our neural network, accelerated by the massive parallelism of GPU, is substantially less sensitive to circuit size. This advantage is particularly crucial in the current era of rapidly increasing circuit complexity.

C. Comparison with Other ML Methods

To further validate the superiority of the proposed model, we conducted a comparative experiment against other graph-based machine learning methods. Graph Convolutional Network (GCN) [23] and GraphSAGE [24] are two common baselines in graph learning tasks. We implemented GCN and GraphSAGE models with the same number and size of graph aggregation layers, as well as MLP structures identical to those of our Graph-SRAM model. Furthermore, to demonstrate the importance of incorporating RC models of the interconnect, we also implemented an ablation variant of our model, Graph-SRAM_wo, which only constructs graphs from the transistors within standard cells.

All four models were trained and tested using identical datasets and procedures. The error distributions of their pre-

dictions on the entire test set are plotted in Fig. 6. The results show that our Graph-SRAM demonstrates the best performance, with the smallest average MARE across all five metrics, and its peak errors are also reduced. This is attributed to the attention mechanism, which enhances the graph aggregation capability to capture critical paths and device interactions, combined with the CNN-based waveform extractor that effectively learns local patterns and temporal dependencies from the input waveform sequences.

In addition, the comparison with Graph-SRAM_wo confirms the benefit of modeling the RC parasitics of interconnect and fusing them with the standard cell graph. This operation leads to an error reduction of 18%, 20%, and 44% for path waveform, read delay, and write delay, respectively. It also demonstrates the significant impact of interconnects on the timing characteristics of SRAMs.

V. CONCLUSION

This paper presents Graph-SRAM, a novel graph learning framework for efficient and accurate waveform simulation in SRAM circuits. Our approach uniquely models activated cells and their interconnect parasitics as a unified graph, embedding global features via switch-level simulation to adeptly manage memory cell behaviors. When tested on SRAMs of various sizes, the model demonstrates its effectiveness by accelerating simulations $6905.32\times$ on average over HSPICE, while concurrently achieving SPICE-level accuracy with MARE values of 4.28% (waveform), 1.90% (read delay), and 2.06% (write delay).

REFERENCES

- [1] M. Qazi, M. Tikekar, L. Dolecek, D. Shah, and A. P. Chandrakasan, "Technique for efficient evaluation of sram timing failure," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 8, pp. 1558–1562, 2013.
- [2] J. Zhai, C. Yan, S.-G. Wang, and D. Zhou, "An efficient bayesian yield estimation method for high dimensional and high sigma sram circuits," in *Proceedings of the 55th Annual Design Automation Conference, DAC '18*, (New York, NY, USA), Association for Computing Machinery, 2018.
- [3] X. Jing and R. Yao, "A fast-simulation model for post-layout sram," in *2007 7th International Conference on ASIC*, pp. 1197–1200, 2007.
- [4] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, "Openram: an open-source memory compiler," in *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD '16*, (New York, NY, USA), Association for Computing Machinery, 2016.
- [5] J. Lee, J. Park, S. Kim, and H. Jeong, "Bayesian learning automated sram circuit design for power and performance optimization," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 12, pp. 4949–4961, 2023.
- [6] C. Ming and B. Na, "An efficient and flexible embedded memory ip compiler," in *2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pp. 268–273, 2012.
- [7] Y. Liu, G. Dai, and W. W. Xing, "Seeking the yield barrier: High-dimensional sram evaluation through optimal manifold," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2023.
- [8] Y. Shen, C. Yan, S.-G. Wang, D. Zhou, and X. Zeng, "An efficient yield estimation method for layouts of high dimensional and high sigma sram arrays," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1723–1728, 2021.
- [9] Z. Zhou and G. Zhang, "The fast simulation model of sram," in *2006 8th International Conference on Solid-State and Integrated Circuit Technology Proceedings*, pp. 1333–1335, 2006.
- [10] P. Zuber, P. Dobrovolny, and M. Miranda, "A holistic approach for statistical sram analysis," in *Proceedings of the 47th Design Automation Conference, DAC '10*, (New York, NY, USA), pp. 717–722, Association for Computing Machinery, 2010.
- [11] Y.-Y. Chen, S.-Y. Huang, and Y.-C. Chang, "Rapid and accurate timing modeling for sram compiler," in *2009 IEEE International Workshop on Memory Technology, Design, and Testing*, pp. 73–76, 2009.
- [12] S. Sangameswaran and S. Yamauchi, "Post-layout parasitic verification methodology for mixed-signal designs using fast-spice simulators," in *2005 IEEE Dallas/CAS Workshop on Architecture, Circuits and Implementation of SOCs*, pp. 211–214, 2005.
- [13] Y. Su, F. Yang, and X. Zeng, "Amor: an efficient aggregating based model order reduction method for many-terminal interconnect circuits," in *Proceedings of the 49th Annual Design Automation Conference, DAC '12*, (New York, NY, USA), pp. 295–300, Association for Computing Machinery, 2012.
- [14] I. Jeon, H. Park, T. Yoon, and H. Jeong, "High efficiency variation-aware sram timing characterization via machine-learning-assisted netlist extraction," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 3, pp. 1391–1395, 2024.
- [15] S. Zhang, F. Yang, D. Zhou, and X. Zeng, "Bayesian methods for the yield optimization of analog and sram circuits," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 440–445, 2020.
- [16] J. Bouhlila, F. Last, R. Buchty, M. Berekovic, and S. Mulhem, "Machine learning for sram stability analysis," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2024.
- [17] P. Cao, G. He, and T. Yang, "Tf-predictor: Transformer-based prerouting path delay prediction framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 7, pp. 2227–2237, 2023.
- [18] 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, DAC '22*, (New York, NY, USA), pp. 1207–1212, Association for Computing Machinery, 2022.
- [19] D. Sánchez, L. Servadei, G. N. Kiprit, R. Wille, and W. Ecker, "A comprehensive survey on electronic design automation and graph neural networks: Theory and applications," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, Feb. 2023.
- [20] Y. Ma, Z. He, W. Li, L. Zhang, and B. Yu, "Understanding graphs in eda: From shallow to deep learning," in *Proceedings of the 2020 International Symposium on Physical Design, ISPD '20*, (New York, NY, USA), pp. 119–126, Association for Computing Machinery, 2020.
- [21] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.
- [22] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.
- [23] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.
- [24] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.