Deep-Learning-Based Pre-Layout Parasitic Capacitance Prediction on SRAM Designs

Shan Shen*, Dingcheng Yang*, Yuyang Xie*, Chunyan Pei*, Wenjian Yu*, Bei Yu+
*Department Computer Science & Technology, BNRist, Tsinghua University, Beijing, China
+Department of Computer Science & Engineering, The Chinese University of Hong Kong, Hong Kong SAR
{shanshen,yu-wj}@tsinghua.edu.cn

ABSTRACT

To achieve higher system energy efficiency, SRAM in SoCs is often customized. The parasitic effects cause notable discrepancies between pre-layout and post-layout circuit simulations, leading to difficulty in converging design parameters and excessive design iterations. Is it possible to well predict the parasitics based on the pre-layout circuit, so as to perform parasitic-aware pre-layout simulation? In this work, we propose a deep-learning-based 2-stage model to accurately predict these parasitics in pre-layout stages. The model combines a Graph Neural Network (GNN) classifier and Multi-Layer Perceptron (MLP) regressors, effectively managing class imbalance of the net parasitics in SRAM circuits. We also employ Focal Loss to mitigate the impact of abundant internal net samples and integrate subcircuit information into the graph to abstract the hierarchical structure of schematics. Experiments on 4 real SRAM designs show that our approach not only surpasses the state-of-the-art model in parasitic prediction by a maximum of 19X reduction of error but also significantly boosts the simulation process by up to 598X speedup.

ACM Reference Format:
Shan Shen*, Dingcheng Yang*, Yuyang Xie*, Chunyan Pei*, Wenjian Yu*, Bei Yu+ . 2024. Deep-Learning-Based Pre-Layout Parasitic Capacitance Prediction on SRAM Designs . In *Great Lakes Symposium on VLSI 2024 (GLSVLSI '24), June 12–14, 2024, Clearwater, FL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649476.3658754

1 INTRODUCTION

With the rapid expansion of intelligent Internet of Things (IoT) devices, contemporary System-on-Chips (SoCs) are increasingly focused on enhancing energy efficiency. This is essential for extending the standby time of battery-powered devices. In order to optimize the energy efficiency of on-chip memory, both academic and industrial researchers have proposed various high-energy-efficiency memory structures. These include near-threshold SRAM [15][2], compute-in-memory SRAM [22][23], and others. SRAM customization involves adjustments in the topology and size of the memory cell, peripheral circuits, timing, and controller design. To ensure the stability of the chip's functionality and its final yield, it is crucial to simulate and evaluate various performance metrics, such as read/write delay, power consumption, and failure probabilities during the design procedure [11]. Once the SRAM performance falls short of expectations, significant time and labor are required for iterative design modifications. This process adds complexity to customizing the SRAM IP and results in prolonged design cycles.

In traditional design workflows, designers proceed with circuit design and optimization based on pre-layout simulations. They then perform verification using post-layout simulations after completing the layout drawing. However, with advanced technologies adopting smaller transistor sizes and lower operating voltages, there's a notable decrease in transistor driving ability. Consequently, the parasitic effect becomes too significant to be overlooked [24]. This disparity leads to a substantial gap between pre-layout and post-layout simulation results, making it challenging to ensure the final circuit performance. Regrettably, the back-end design of customized SRAM is an arduous and time-intensive process. This is due to two primary reasons: (1) the high density of transistors often leads to violations of design rules; (2) the routing process is complicated, as it involves managing a substantial number of data wires within a constrained area. Consequently, even minor alterations to the schematic can necessitate extensive modifications in the layout.

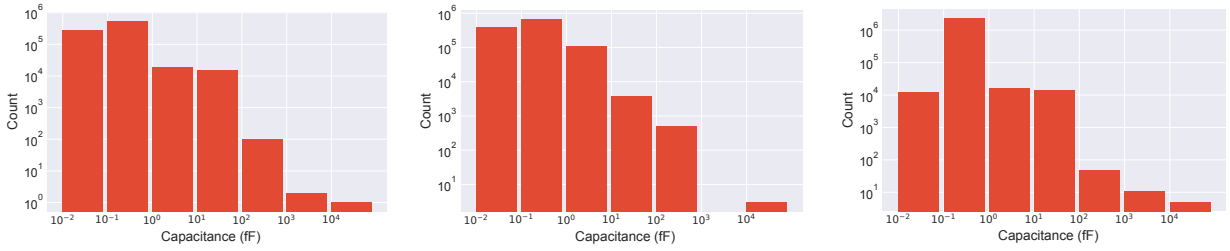
Recently, numerous studies [3, 7, 9, 12, 13, 17, 21] have employed machine learning as a potent tool for predicting parasitic effects in electronic design. However, training accurate ML-based models is often hindered by the class imbalance of parasitic capacitance. As illustrated in Fig. 1, the distribution of net capacitance shows a prominent imbalance, with values ranging from 0.01 fF to 100 pF. There are over 10^6 nets in the second bin of SP8192W SRAM, predominantly internal connections in memory cells. This imbalance presents two problems: (1) the training process becomes inefficient, as the majority of nets are "easy negatives" that offer little to no valuable learning signal; (2) these easy negatives can dominate the training process, potentially leading to the development of ineffective models. In this work, we aim to address this challenge and enhance the accuracy of a deep-learning-based model for parasitic prediction. The contributions of this work are outlined as follows.

- We propose a unique 2-stage model, consisting of a Graph Neural Network (GNN) classifier and multiple Multi-Layer Perceptron (MLP) regressors, and the corresponding training strategy. We implement Focal Loss [8] as the loss function of the classifier to further reduce the overwhelming effect of easy negatives. Subcircuit information is also integrated into the graph, effectively mirroring the hierarchical structure found in schematics. Our experimental results demonstrate that the 2-stage model achieves an accuracy improvement ranging from 2.5X to 19X

This work was partially supported by the National Science and Technology Major Project (2021ZD0114703), and the National Natural Science Foundation of China (NSFC) under grant No. 62204141 and 62090025. W. Yu is the corresponding author.
Email: {shanshen, yu-wj}@tsinghua.edu.cn.

GLSVLSI '24, June 12–14, 2024, Clearwater, FL, USA

ACM ISBN 979-8-4007-0605-9/24/06
https://doi.org/10.1145/3649476.3658754



(a) (b) (c)

Figure 1: Distributions of the parasitic net capacitance of (a) Ultra8T SRAM, (b) Sandwich-RAM [22], and (c) SP8192W SRAM [1].

over the state-of-the-art model [13], while also reducing both training and inference time.

- The proposed method stands out from existing parasitic prediction models that primarily focus on small-scale analog circuits, as it targets large-scale memory circuits. By simulating the schematic netlist with the predicted parasitics, we achieve a significant speedup, up to 586X, compared to the simulations using the post-layout netlist. This approach substantially benefits the large-scale memory circuit design. The versatility of the proposed method also makes it naturally suitable for other downstream tasks. This includes design space exploration and expedited design updates.

The paper is organized as follows. Section 2 reviews the related work in the field and Section 3 introduces some fundamental concepts of GNNs. Section 4 presents the proposed model for net capacitance prediction, and Section 5 describes the over workflow. Section 6 gives experimental results derived from real SRAM designs. Section 7 concludes the whole paper.

2 RELATED WORKS

In recent years, a lot of attention has been paid to machine learning as an effective method for parasitic prediction. Shook *et al.* [17] transformed the front-end netlist (schematic) of analog IP designs into a star topology and used a random forest model to predict the equivalent resistance and capacitance of each net. Another work, named ParaGraph [13], converts circuit schematics into graphs and utilizes graph neural network (GNN) techniques to predict net capacitance and device layout parameters. It leverages a complex aggregation procedure to compute node embedding, which is a combination of graph convolutional network (GCN) [6], GraphSage [5], relation GCN (RGCN) [14], and graph attention network (GAT) [18]. It also includes the ensemble modeling technique to improve the prediction accuracy via training 3 different sub-models for different magnitudes of the capacitance value. Net samples with a ground truth larger than the maximum predicted value of the sub-model are ignored during training, which increases prediction accuracy within a specified range within each model. The sub-model with larger capacitance prediction is more preferred than that with small capacitance. Li *et al* [7] adopted a ParaGraph-like model to predict the parasitics and guide optimization of the voltage-controlled oscillator (VCO). Liu *et al.* [9] proposed an improved surrogate performance model using parasitic graph embeddings generated by ParaGraph [13]. Then the surrogate model was integrated into a Bayesian optimization workflow to automate transistor sizing.

Machine learning-based capacitance extraction is also studied for the scenario of monolithic 3D (M3D) IC design. Pentapati *et al.* [12] proposed a regression model based on augmented decision tree learning to better predict 3D net parasitics. In another scenario, Yang *et al.* [21] proposed a convolutional neural network capacitance model (CNN-Cap) for the pattern-matching-based capacitance extraction. The method is able to accurately compute the capacitances of 2D patterns with a variable number of conductors.

It should be pointed out that although [7, 9, 13, 17] are for predicting net capacitances at the pre-layout stage, they are all focused on analog circuits. Compared to the analog circuit, large-scale high-density memory circuits suffer from a more severe imbalance of training data. Besides, authors in [12] and [3], etc. assume the layout is partially ready, which is not purely based on the pre-layout schematic. In this work, we focus on predicting the net capacitances at the pre-layout stage to expedite the customized SRAM design.

3 PRELIMINARIES

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a structure used to represent entities and their relations. It consists of two sets, the set of nodes $\mathcal{V}$ (also called vertices) and the set of edges $\mathcal{E}$ (also called arcs). Each node is associated with a vector of features $x_v = (x_1, ..., x_d)$ with dimension d. The $n = |\mathcal{V}|$ node features form a matrix $X \in \mathbb{R}^{n \times d}$. An edge $(u, v) \in \mathcal{E}$, represented as $e_{u,v}$, connecting a pair of nodes u and v indicates that there is a relation between them. The edge can also have a feature vector $x_e = (x_1, ..., x_c)$ with dimension c and $m = |\mathcal{E}|$ features form a matrix $X^e \in \mathbb{R}^{m \times c}$. The neighborhood of a node $\mathcal{N}(v)$ is defined as $\mathcal{N}(v) = \{u \in \mathcal{V} \mid (u, v) \in \mathcal{E}\}$. Graphs can be either homogeneous or heterogeneous. In a homogeneous graph, all the nodes represent instances of the same type and all the edges represent relations of the same type. In contrast, in a heterogeneous graph, the nodes and edges can be of different types.

GNN is a kind of neural network that operates directly on data structured as graphs, without losing structural and feature information [4]. Having an input graph, a GNN aims to learn the embedding vectors per node, defined as $h_u, \forall u \in \mathcal{V}$, which encodes the neighborhood information of each node [20]. The message passing between nodes is assumed as the most generic GNN layer [10]. Given a graph structure, message passing updates the edge embeddings $h^e_{(u,v)}$ with

$$h^e_{(u,v)} = \phi(h_u, h_v, x^e_{(u,v)}), \tag{1}$$

where $\phi(\cdot)$ is an arbitrary, non-linear, differentiable function that aggregates its inputs, and $x^e_{(u,v)} \in \mathbb{R}^c$ is the initial edge feature

vector. After the edge embedding is obtained, and defining $x_u \in \mathbb{R}^d$ as the feature vector for the starting node, the node representation is updated by

$$h_u^{'} = \phi^{'}(h_u, \sum_{v \in N(u)} h^e_{(u,v)}, x_u). \quad (2)$$

The graph embeddings learned by GNNs can be used as inputs to other ML models for building an end-to-end framework. There are three levels of tasks for such a framework: node, edge, and graph [20]. In the node-level task, the regression or classification problem of the nodes is of concern.

4 PARASITIC CAPACITANCE PREDICTION

A 2-stage deep-learning model is proposed in this section. It contains a GNN-based classifier and 5 MLP regressors. We first introduce the conversion of schematics, then the feature extraction, and last the model architecture and training strategy.

4.1 Conversion of Circuit Netlist to Graph

In order to reflect the modularization in circuit design, the schematic netlist will be modeled as a heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in this work. A node in the graph corresponds to a net, a device, or a subcircuit instance in the circuit. Fig. 2 shows an example of a buffer containing three types of node sets $\mathcal{V} = \mathcal{V}^{\text{NET}} \cup \mathcal{V}^{\text{DEV}} \cup \mathcal{V}^{\text{SUB}}$. A green circle represents a net ($v \in \mathcal{V}^{\text{NET}}$), connecting to devices; an orange square represents an instance of the transistor device ($v \in \mathcal{V}^{\text{DEV}}$), which can also be other types of devices such as a capacitor, a resistor, and a diode; a blue triangle represents an instance of the subcircuit ($v \in \mathcal{V}^{\text{SUB}}$), comprised of multiple nets and device instances. We set all edges in the graph as undirected. The advantage of the undirected graph is that feature information can be transferred between different nodes in a shallow network. Compared to the graphs only comprising nodes representing nets and devices in [13], the node of the subcircuit instance in this work can reflect the hierarchical structure in schematics and generalize the local features.

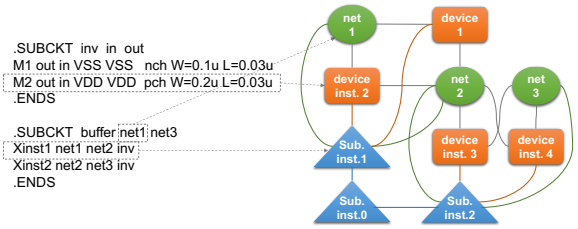


Figure 2: Example of converting a circuit to a graph.

4.2 Acquisition of Training Data

In the post-layout netlist (SPF files) generated by RC extraction, the net parasitics form a complex π-type RC network. As the net capacitance is of concern, we generate the lumped sum of capacitance (C_{eff}) for each net from the post-layout netlist and use it as the ground-truth label for training the 2-stage model.

We extract three types of feature vectors from the schematic netlist. The definitions of different feature elements are listed in Table 1. Device nodes need to extract different features according to their device types. Compared to existing works, we collect more features for different nodes. For example, feature elements of a transistor

Table 1: Feature definitions of nets, device instances, and subcircuit instances

Type	Notation	Definition	Index
Net	N_{mos}	# of connected transistors	0
	N_g	# of connected gate terminals	1
	N_{sd}	# of connected source/drain terminals	2
	N_b	# of connected base terminals	3
	W_{tot}	Total width of connected transistor	4
	L_{tot}	Total length of connected transistor	5
	N_{cap}	# of connected capacitors	6
	Lr_{tot}	Total length of connected capacitors	7
	Nr_{tot}	Total # of connected capacitor fingers	8
	N_{res}	# of connected resistors	9
	W_{tot_res}	Total width of connected resistors	10
	L_{tot_res}	Total length of connected resistors	11
	N_{port}	# of connected ports	12
Device Instance	M_{mos}	Multiplier of transistors	0
	L	Length of the transistor	1
	W	Width of the transistor	2
	M_{res}	Multiplier of connected resistors	3
	L_{res}	Length of resistor	4
	W_{res}	Width of resistor	5
	M_{cap}	Multiplier of connected capacitor	6
	Lr	Length of capacitor	7
	Nr	# of capacitor fingers	8
	N_p	# of ports in the device instance	9
	T	Type code of the device instance	10
Subcircuit Instance	N_{port}	# of ports in the device instance	0
	N_d	# of ports in the device instance	1
	N_n	# of nets in the subcircuit instance	2
	Lvl	Hierarchy level of the instance	3

device include the multiplier (M_{mos}), channel length (L), width (W), etc., while other feature elements (such as M_{res}, L_{res}, and W_{res}) are zeros. All features are normalized by the maximum value in the dataset to achieve better numerical stability.

4.3 Two-Stage Model Building

The distribution of C_{eff} is extremely imbalanced (Fig. 1). The large number of nets with small capacitance, called easy negatives, results in the minority samples being prone to be mispredicted. However, those nets with large C_{eff} are usually important ports or clock nets, which are likely to affect the timing analysis results. Therefore, we divide the training data (net nodes) into 5 categories and label them with $t \in \mathcal{T} = \{0, 1, 2, 3, 4\}$ according to the magnitude of their parasitic capacitance, i.e., {(0.01 fF, 0.1 fF], (0.1 fF, 1 fF], (1 fF, 10 fF], (10 fF,100 fF], (100 fF,∞)}.

GNN is more suitable for classification tasks. In order to do the capacitance regression with GNN, we build a 2-stage model based on GNN and multilayer perceptron (MLP). The model is mainly divided into the net classification and the net capacitance regression, as shown in Fig. 3. Feature vectors (different dimensions) of three types of nodes are projected to a common space through 3 different projectors individually so that we can easily transform a heterogeneous graph into a homogeneous one. It is convenient for subsequent training of different GNN models. Next, the projected feature vectors are input into the GNN+MLP model to classify the net nodes into different categories. Embeddings of net nodes generated by the GNN model will be fed into a 3-layer MLP to predict the label t of a net node. Note that only the net nodes have labels and their embeddings are retained at this time, while that of the other two types of nodes will be masked. To reduce both the training and inference time, our proposed method leverages simple GNN models but uses a delicate training strategy and loss function.

In the 2nd stage, capacitance regression is performed for each category individually. The input of the regression model is a concatenated vector containing node embeddings and the original net feature vector. The target vector is the effective net capacitance C_{eff}.

The training flow is also divided into 2 stages. We first use feature vectors and labels of net nodes to train and validate the classifier. Once the classifier is obtained, according to the classification result, net nodes in the training set are grouped into different sub-sets and are used to train their own regressors with the corresponding targets C_{eff}, respectively. The test set is invisible during the whole training process.

In the 1st stage, we adopt Focal Loss [8] with a weight factor $\alpha \in [0, 1]$ which applies a modulating term to the cross-entropy loss in order to focus learning on hard examples and down-weight the numerous easy negatives.

$$\mathcal{L} = \sum_{v \in \mathcal{V}^{\mathrm{NET}}} FL_v = \sum_{v \in \mathcal{V}^{\mathrm{NET}}} -\alpha_t (1 - p_v(t))^{\gamma} \log(p_v(t)), \tag{3}$$

where $\gamma \geq 0$ is a tunable focusing parameter, $p_v(t) \in [0, 1]$ is the model's estimated probability for the net node $v \in \mathcal{V}^{\mathrm{NET}}$ to be in class $t \in \mathcal{T}$. Here we set α to be

$$\alpha_t = \frac{1}{f_t}, \tag{4}$$

where f_t is the proportion of class t in the entire data samples. It can also be set to other values according to the classification results to increase the contribution of categories with fewer data to the total loss function.

In the 2nd stage, we use the mean value of the squared percentage error as the loss function in the regression training, i.e.

$$\mathcal{L}_t = \frac{1}{N_t} \sum_{v \in \mathcal{V}^{\mathrm{NET}}|t} \left(\frac{y_v - \hat{y_v}}{y_v} \right)^2, \tag{5}$$

where symbol y denotes the target value and $\hat{y}$ the predicted value. N_t is the number of nets with predicted label t.

5 OVERALL WORKFLOW

The workflow of the proposed method is illustrated in Fig. 4. Training data are collected from the pre-layout schematic and the post-layout netlist by matching net names. During training, the input of the 2-stage model includes feature matrices from 3 types of nodes, a graph converted from the schematic, a class label vector, and a net capacitance vector. The output of the model is the predicted C_{eff}. The regressors of stage 2 can be trained in a parallel manner to further

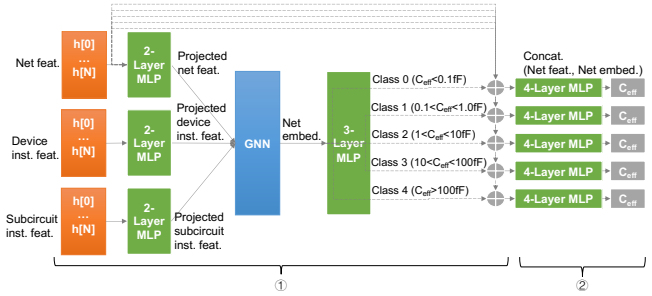


Figure 3: Structure of the proposed 2-stage model including net classification stage and capacitance regression stage.

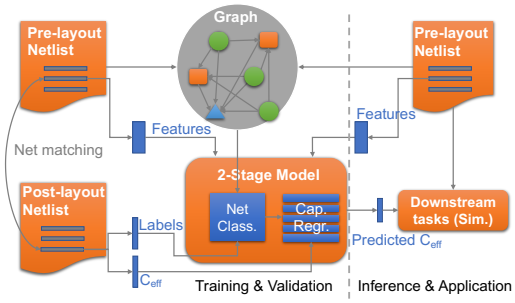


Figure 4: The overall workflow of the proposed method.

speed up the model training. During the inference, the design's graph and the features of nets are fed into the model. The predicted net capacitance from the model is back-annotated into a netlist for downstream tasks.

6 EXPERIMENTAL RESULTS

Experiments are conducted to show the effectiveness of the proposed method with 4 SRAM designs. All ML models are implemented based on DGL library [19] written in PyTorch. The post-layout netlists with full parasitics are extracted by StarRC. All experiments are run on a server with 40 Intel Xeon Silver CPUs with 128GB memory.

6.1 Dataset

We prepare the full schematic netlists and the post-layout SPF netlists to extract the feature vectors and net capacitance respectively. Table 2 summarizes the SRAM design examples utilized by this work (net parasitic capacitance distributions are illustrated in Fig. 1). All designs are under 28nm CMOS technology. SSRAM [15] is a small design with high-energy efficiency with a timing-speculation technique. The Sandwich-RAM [22] is made up of half of digital circuits for computing and half of memory arrays, forming a sandwich-like structure. Ultra8T SRAM [16] is a multi-voltage design with a wide range of operation voltages, which contains analog circuits. SP8192W SRAM is based on a single port 6T cell structure generated by the SRAM compiler [1] with tremendously high density. We leverage the full neighbor sampling method provided by DGL, which randomly splits the test set from the original large graph and returns a new subgraph. This ensures the invisibility of nodes in the test set during model training. The train/validation/test set split ratio for each design is 0.6/0.2/0.2.

Table 2: SRAM designs used by this work.

SRAM Designs	SSRAM	Ultra8T	Sandwich	SP8192W
# of Nets	19902	861842	1160940	2342588
# of Device Inst.	57417	2325092	2665422	6984821
# of Subcircuit Inst.	9965	315466	428498	39885
Total # of Nodes	87284	3502400	4254860	9367294
# of Edges	134926	13392268	13254854	32009072

6.2 Model Settings

In the 1st stage, we adopt three mainstream GNN structures, including a 3-layer graph attention network (GAT) [18], a 3-layer graph convolutional networks (GCN) [6], and 2-layer GraphSAGE [5], in net classification. GraphSAGE can be implemented with different types of aggregators in (1) and (2), and we use 'mean' and 'pool' in

our comparison. We set all layer widths to 64. The feature projection MLP has 2 linear layers and the classification MLP has 3 linear layers. The activation function is ReLU and the dropout is 0.1 for all layers. The batch normalization (BN) is turned on. The GAT-based classifier needs to turn on the layer normalization and turn off the BN to get reasonable accuracy. We use a cosine annealing schedule to adjust the learning rate based on the number of executed epochs. The learning rate ranges from 1E-3 to 1E-4. The weight decay parameter is set to 5E-4. The performance metrics include accuracy, and the F1 macro score (the best value is 1.0 for all metrics). F1 macro is the unweighted mean of the F1 score of each class. The training process is run 10 times for each model and the performance metrics are averaged from the best epochs.

In the 2nd stage, each regressor is a 4-layer MLP with hidden layer widths {128, 128, 64}. The input width is the sum of the node embedding width 64 and the original net feature width 13 while the output width is 1. The activation function is ReLU and the dropout is 0.5. The learning rate is set to 1E-3 and the weight decay is set to 5E-4. The following mean absolute percentage error (MAPE) is used as the performance metric:

$$\text{MAPE} = \frac{100\%}{N} \sum_{i}^{N} \left| \frac{y_i - \hat{y}_i}{y_i} \right|. \tag{6}$$

We also implement ParaGraph [13]. ParaGraph is comprised of 3 sub-models to predict the net capacitance ranging from (0.01fF, 1fF], (1fF, 10fF], (10fF, 10pF], respectively. Each submodel has a 32-dimension width, a 5-layer GNN model, and a 4-layer MLP regressor.

6.3 Classification and Regression Results

Table 3 shows the classification results using different GNN models. The GraphSAGE-based classifier has the best accuracy across all design cases. The attention-based classifier has relatively lower F1 macro scores than others. Besides, SP8192W SRAM has special characteristics since all classifiers have reduced scores in this case. It can be explained by the imbalanced C_{eff} distribution in Fig. 1, where the number of easy negatives from class 1 is over 100X larger than that of other classes.

In Fig. 5, we compare the classification results with different training strategies. After changing the loss function from cross entropy to Focal loss, the F1 macro scores are improved significantly. Moreover, by introducing the subcircuit instance nodes in the graph, the hierarchical information improves the performance of the GAT-based, GCN, GraphSAGE_mean-based classifiers, but slightly degenerates that of the GraphSAGE_pool-based model. We also find the existence of subcircuit nodes helps GCN and GraphSAGE_mean to converge more quickly during training.

Table 3 also compares the accuracy of different ML-based regression models. ParaGraph's prediction errors are listed in the last column, as it does not incorporate the net classification stage. ParaGraph has the worst prediction accuracy, over 30% MAPEs for the first 3 design cases. This is because it fails to solve the class imbalance in the dataset. The 'None' row from each design case represents the 5 pure regression models without introducing any classification and feature projection. They are trained individually according to the ground truth labels of samples. The input of the regressors is just the original features of net nodes. Compared to the 'None' model, GNN-based models have lower predicted errors across all net classes. For the GraphSAGE model, using a 'mean' aggregator is a good choice in our experiments and achieves better accuracy. The GCN-base and the GraphSAGE_mean-based regression models have similar accuracy across all design cases. The minimum and maximum MAPE reductions of the proposed 2-stage models against ParaGraph are 2.5X in SP8192W and 19X in Ultra8T. Notice that for SP8192W SRAM, the MAPE increases for all models when predicting C_{eff} of the nets in class 0 and class 2. This is due to a lot of false positives residing in these 2 classes (see the low F1 macro scores) as noise.

6.4 Other Comparisons

Fig. 6 shows the power consumption collected from the pre-layout simulation, the post-layout simulation, and the proposed method. With the predicted C_{eff}, the performance error of the pre-layout simulation is largely reduced from 57.24% to 17.77% on average. Moreover, since our method only uses schematics, the maximum simulation speedup reaches 586X for Ultra8T while the minimum speedup is 19.66X for SSRAM compared to the post-layout simulation.

Table 4 further lists the scales of different models, and the training/inference time. ParaGraph [13] exhibits the largest training and inference time due to the complicated aggregation function that requires an attention operation for each edge type. In general, the proposed GraphSAGE_mean-based model is the most efficient model. The proposed models have a larger number of trainable parameters than ParaGraph due to the existence of the 5 regressors.

7 CONCLUSION

This paper presents a novel method to train a 2-stage model based on GNN and MLP for predicting parasitic capacitances in SRAM

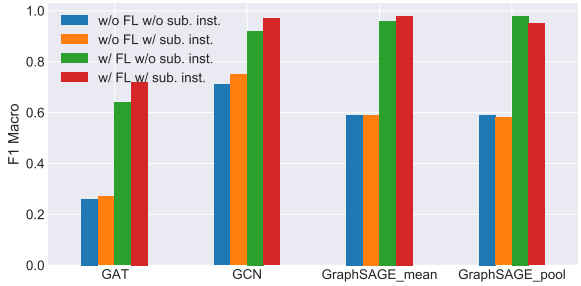


Figure 5: Performance comparison of the GNN-based classifiers trained by different strategies.

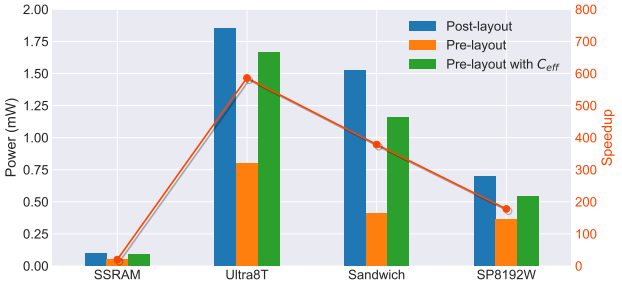


Figure 6: Simulated power consumption using different netlists, and the speedup of the proposed method.

Table 3: The classification accuracy, F1 macro, and mean absolute percentage error (MAPE) of the regressors with different GNN classifiers on the test set. (The figures meaning the best performance are highlighted)

Design Case	Choice of Classifier	Class Acc.	F1 Macro	Class 0	Class 1	Class 2	Class 3	Class 4	All tested nets
SSRAM	ParaGraph[13]	-	-	-	-	-	-	-	33.53%
	None	-	-	21.73%	5.53%	28.84%	6.59%	-	6.95%
	GAT	93.34%	0.83	26.15%	**0.88%**	26.37%	1.68%	-	4.30%
	GCN	95.38%	0.88	15.58%	2.22%	19.00%	5.66%	-	4.05%
	GraphSAGE_mean	97.98%	0.93	24.19%	1.37%	**14.09%**	6.35%	-	**3.13%**
	GraphSAGE_pool	**99.22%**	**0.97**	**14.11%**	8.70%	31.94%	**1.34%**	-	9.66%
Ultra8T	ParaGraph[13]	-	-	-	-	-	-	-	32.01%
	None	-	-	4.12%	11.59%	28.81%	20.79%	9.49%	9.78%
	GAT	96.61%	0.90	2.57%	**1.30%**	17.10%	5.25%	45.07%	2.67%
	GCN	98.36%	0.94	1.15%	1.59%	**6.11%**	4.89%	9.10%	**1.68%**
	GraphSAGE_mean	**99.91%**	**0.99**	**0.95%**	1.87%	6.77%	**3.56%**	**5.63%**	1.72%
	GraphSAGE_pool	98.29%	0.96	3.10%	9.89%	14.07%	5.92%	10.03%	7.74%
Sandwich	ParaGraph[13]	-	-	-	-	-	-	-	34.98%
	None	-	-	23.57%	25.05%	29.01%	55.78%	8.20%	25.07%
	GAT	87.04%	0.72	15.92%	10.08%	20.43%	30.08%	35.28%	13.60%
	GCN	97.99%	0.97	3.83%	6.98%	10.12%	10.46%	**6.71%**	6.25%
	GraphSAGE_mean	**98.72%**	**0.98**	**3.11%**	**5.96%**	**8.56%**	**8.21%**	8.49%	**5.32%**
	GraphSAGE_pool	96.33%	0.95	3.84%	7.91%	10.99%	14.07%	8.06%	6.83%
SP8192W	ParaGraph[13]	-	-	-	-	-	-	-	4.61%
	None	-	-	26.68%	1.14%	38.27%	6.74%	55.75%	1.62%
	GAT	99.17%	0.72	33.34%	0.36%	25.95%	2.71%	9.34%	0.87%
	GCN	99.22%	0.76	26.32%	**0.26%**	22.70%	2.74%	21.18%	**0.81%**
	GraphSAGE_mean	**99.72%**	**0.88**	**20.59%**	0.73%	**12.36%**	**2.34%**	**3.99%**	1.04%
	GraphSAGE_pool	99.40%	0.82	24.38%	1.31%	26.66%	5.33%	6.10%	1.82%

Table 4: Storage and time overhead of different GNN-based models.

Info.	ParaGraph[13]	GAT	GCN	GraphSAGE_mean	GraphSAGE_pool
# of params	141,987	212,362	212,618	216,650	224,970
train. time(h)	21.59	16.43	13.41	7.65	15.94
infer. time(s)	27.10	17.28	5.58	4.07	6.08

designs. This model well handles the class imbalance of net parasitics in SRAMs, and thus outperforms the existing state-of-the-art model. In the future, the proposed method will be extended to complete RC prediction and integrated into circuit optimization algorithms of energy-efficient SRAM design.

REFERENCES

[1] ARM. 2023. Artisan Embedded Memory IP. https://www.arm.com/en/products/silicon-ip-physical/embedded-memory

[2] Yung-Chen Chien and Jinn-Shyan Wang. 2018. A 0.2 V 32-Kb 10T SRAM with 41 nW standby power for IoT applications. *IEEE Trans. Circuits Syst. I* 65, 8 (2018), 2443–2454.

[3] Weibing Gong, Wenjian Yu, Yongqiang Lü, Qiming Tang, Qiang Zhou, and Yici Cai. 2010. A parasitic extraction method of VLSI interconnects for pre-route timing analysis. In *Proc. Int. Conf. on Commun., Circuits and Syst. (ICCCAS)*. 871–875.

[4] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *Proc. Int. Joint Conf. on Neural Netw. (IJCNN)*. 729–734.

[5] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in Neural Information Process. Syst.* 30 (2017).

[6] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[7] Chenfeng Li, Dezhong Hu, and Xiaoyan Zhang. 2023. Pre-Layout Parasitic-Aware Design Optimizing for RF Circuits Using Graph Neural Network. *Electronics* 12, 2 (2023), 465.

[8] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proc. Int. Conf. on Comput. Vision (ICCV)*. 2980–2988.

[9] Mingjie Liu, Walker J Turner, George F Kokai, Brucek Khailany, David Z Pan, and Haoxing Ren. 2021. Parasitic-aware analog circuit sizing with graph neural networks and Bayesian optimization. In *Proc. DATE*. 1372–1377.

[10] Daniela Sánchez Lopera, Lorenzo Servadei, Gamze Naz Kiprit, Souvik Hazra, Robert Wille, and Wolfgang Ecker. 2021. A survey of graph neural networks for electronic design automation. In *Proc. MLCAD*. 1–6.

[11] Saibal Mukhopadhyay, Hamid Mahmoodi, and Kaushik Roy. 2005. Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 24, 12 (2005), 1859–1880.

[12] Sai Surya Kiran Pentapati, Bon Woong Ku, and Sung Kyu Lim. 2021. ML-based wire RC prediction in monolithic 3D ICs with an application to full-chip optimization. In *Proc. ISPD*. 75–82.

[13] Haoxing Ren, George F Kokai, Walker J Turner, and Ting-Sheng Ku. 2020. ParaGraph: Layout parasitics and device parameter prediction using graph neural networks. In *Proc. DAC*. 1–6.

[14] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. June 3–7, 2018. Modeling relational data with graph convolutional networks. In *Proc. European Semantic Web Conference (ESWC), Heraklion, Crete, Greece*. 593–607.

[15] Shan Shen, Tianxiang Shao, Xiaojing Shang, Yichen Guo, Ming Ling, Jun Yang, and Longxing Shi. 2019. TS cache: A fast cache with timing-speculation mechanism under low supply voltages. *IEEE Trans. VLSI Syst.* 28, 1 (2019), 252–262.

[16] Shan Shen, Hao Xu, Yongliang Zhou, Ming Ling, and Wenjian Yu. 2023. Ultra8T: A Sub-Threshold 8T SRAM with Leakage Detection. *arXiv preprint arXiv:2306.08936* (2023).

[17] Brett Shook, Prateek Bhansali, Chandramouli Kashyap, Chirayu Amin, and Siddhartha Joshi. 2020. MLParest: Machine learning based parasitic estimation for custom circuit design. In *Proc. DAC*. 1–6.

[18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[19] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, et al. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315* (2019).

[20] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 32, 1 (2020), 4–24.

[21] Dingcheng Yang, Haoyuan Li, Wenjian Yu, Yuanbo Guo, and Wenjie Liang. 2023. CNN-Cap: Effective Convolutional Neural Network-based Capacitance Models for Interconnect Capacitance Extraction. *ACM Transactions on Design Automation of Electronic Systems* 28, 4 (2023), 1–22.

[22] Jun Yang, Yuyao Kong, Zhen Wang, Yan Liu, Bo Wang, Shouyi Yin, and Longxin Shi. 2019. 24.4 sandwich-RAM: An energy-efficient in-memory BWN architecture with pulse-width modulation. In *Proc. Int. Solid-State Circuits Conf. (ISSCC)*. 394–396.

[23] Chengshuo Yu, Taegeun Yoo, Kevin Tshun Chuan Chai, Tony Tae-Hyoung Kim, and Bongjin Kim. 2022. A 65-nm 8T SRAM compute-in-memory macro with column ADCs for processing neural networks. *IEEE J. Solid-State Circuits* 57, 11 (2022), 3466–3476.

[24] Wenjian Yu and Xiren Wang. 2014. *Advanced field-solver techniques for RC extraction of integrated circuits*. Springer.