ParaGraph: Layout Parasitics and Device Parameter Prediction using Graph Neural Networks

Haoxing Ren
NVIDIA
Austin, TX, USA
haoxingr@nvidia.com

George F. Kokai
NVIDIA
Santa Clara, CA, USA
gkokai@nvidia.com

Walker J. Turner
NVIDIA
Durham, NC, USA
wturner@nvidia.com

Ting-Sheng Ku
NVIDIA
Santa Clara, CA, USA
tku@nvidia.com

***Abstract*—Layout-dependent parasitics and device parameters significantly impact integrated circuit performance and are often the cause of slow convergences between schematic and layout designs. Circuit designers typically estimate parasitics from past experience, resulting in variability between designers and the potential for inaccuracies. In this paper, we present ParaGraph: a graph neural network model to predict net parasitics and device parameters by converting circuit schematics into graphs and leveraging key modeling techniques based on GraphSage, Relation GCN and Graph Attention Networks. Furthermore, the use of ensemble modeling increases model accuracy over a large range of prediction values. Trained on a large dataset of industrial circuits, the model achieves an average prediction R^2 of 0.772 (110% better than XGBoost) and reduces average simulation errors from over 100% with designer's estimation to less than 10%.**

I. Introduction

The dependency of net parasitics and physical device parameters on circuit layout is becoming increasingly significant within newer process nodes, where accurate evaluation of circuit performance can only be achieved after layout is complete. This paper proposes a technique to predict net parasitics and layout-dependent device parameters based only on the circuit schematic. There are two main benefits to accurately predicting these variables before starting the layout process. First, inclusion of net parasitics and layout-dependent device parameters is likely to degrade circuit performance. This typically results in the designer performing an iterative process of adjusting device sizes post-layout to dial in performance, a process that can take several iterations for the circuit and layout modifications to converge. This becomes significantly more labor intensive due to the increasing complexity of DRC rules within newer technology nodes, where a single layout iteration can take days or weeks. Second, parasitic and device parameters are important factors to include within parasitic-aware optimization techniques as in [1]. An accurate predictor can help optimization engines find design points that represent the true post-layout optimum.

Previous work has predicted layout parasitics and device parameters within sub-micron technologies such as 130nm and 90nm process nodes [2]. Their approach uses layout construction to build estimated layouts of each device and calculate device geometries accordingly. They use a linear regression model to estimate net capacitances based on the number and size of devices connected to each net. The accuracy of this approach largely depends on the estimation of maximal transistor series (MTS), a term relating whether transistors are sharing source or drain diffusion. There are two drawbacks of this approach. First, the MTS estimation is difficult to perform prior to layout and requires designers to manually identify all MTS groupings within the circuit. Second, the layout construction approach uses simplified design rules targeting standard cells. These circuits are comparatively smaller than most analog and mixed-signal designs where layout cannot be easily constructed with simplified rules.

This material is based upon work supported by DARPA under Contract No. HR0011-18-3-0010. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the U.S. Government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited)."

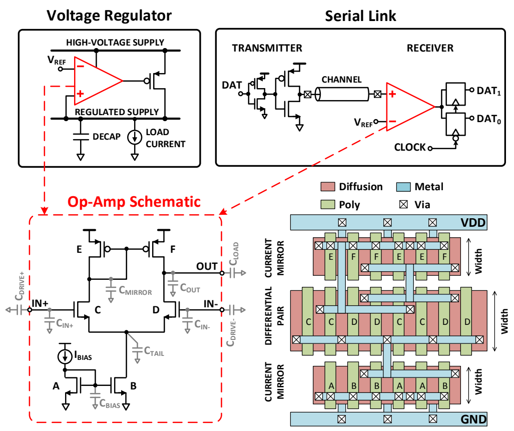


Fig. 1: Circuits sharing a common structure (op-amp) and its layout

There are numerous circuit topologies and transistor configurations that commonly recur within various applications. In order to improve yield and performance, these structures adhere to specific layout configurations and methodologies. This produces net parasitics and device parameters that are a function of device sizing within transistor configurations on the micro and macro scales. This is seen in Figure 1, where an operational amplifier (op-amp) can be used for both voltage regulation and signal amplification, yet the structure layout uses similar approaches in both scenarios. This is also seen in the MTS identification process used in the previous work, where circuit structures are manually identified to determine how diffusion will be shared between transistors and thus affect device parasitics. While heuristic-based "rule of thumb" estimations are often used in experienced design teams, the estimation accuracy is not guaranteed and can vary between cases and individual designers. Instead, we propose a Machine Learning (ML) prediction method based on the inherent graph structure of a circuit schematic. By representing circuit structures as graphs, information regarding net parasitics and device parameters within circuits sharing similar structures can be leveraged in making predictions during the initial design phase. Graph neural networks (GNNs) have been proposed for learning graph structures and predicting useful graph information previously [3] [4] [5] [6]. One relevant example is in the EDA domain, where graph convolutional networks (GCN) were used to accurately predict gate testability within digital circuits [7].

The contributions of this work are highlighted below:

1. We apply GNNs for use in predicting layout parasitics and device parameters, which is the first instance of this to the best of our knowledge. The prediction model is evaluated on a large dataset of analog and mixed-signal circuits from industry, where average spice

simulation error is reduced from over 100% with designer intuition-based prediction to less than 10% with the GNN predictions.

2. We designed our own graph network (ParaGraph) that incorporates key ideas from state-of-the-art GNN models such as GraphSage [4], Relational GCN [5] and Graph Attention Network [6] while adapting out network to the circuit prediction problem domain as a heterogeneous graph with heterogeneous nodes and edge types. The results show a 10%-110% improvement over state-of-art GNN models and XGBoost.

3. We introduce an ensemble modeling method to improve prediction accuracy for parasitic capacitances over a large range from 0.01fF to 10pf.

The paper is organized as follows. Section II defines the layout parasitics and device parameters to be predicted, the input features and the graph construction process. Section III introduces various graph neural networks and presents the ParaGraph model. Section IV introduces the ensemble modeling method to handle a large range of prediction values. Section V presents experimental results based on training on a large set of industrial circuits.

II. Problem Formulation

This work aims to predict both net parasitics and physical device parameters pre-layout. The model is trained to predict net parasitics as lumped capacitances. The predicted device parameters include various layout geometry parameters, such as source/drain diffusion areas and perimeters, as well as layout dependent effect (LDE) parameters such as transistor-to-diffusion-edge distances.

A. Layout Parasitics, Device Parameters and Input Features

Layout for a FinFET transistor is illustrated in Figure 2. For each transistor, we need to predict the transistor's geometric and LDE parameters. Geometric parameters include diffusion areas and perimeters for both the source and drain terminals, which differ depending on whether two adjacent devices are sharing the source/drain diffusion area. For example, the source diffusion area (SA) of the active device **A** on the left is twice as large as its drain diffusion area (DA) because its drain diffusion is shared with the active device **B** on the right. LDE parameters can significantly affect device performance within newer process nodes. One of the more important LDE parameters is the length of diffusion (LOD), which measures the distance of a poly gate to the diffusion edge and is correlated to the strain applied within the silicon channel and how it affects channel mobility. For example, in Figure 2, the left-hand LOD for device **A**: LOD-L(A) is smaller than that of device **B**: LOD-L(B). To handle transistors with multiple fingers, LDE parameters are averaged across all parallel fingers.

Net parasitics represent capacitive and resistive impedances introduced by metal interconnects within the layout. The total capacitance on each net is represented as a lumped sum for convenience and is the focus of this work. While the model can be extended to represent via and trace resistances, inclusion of multi-path trace resistances significantly complicates circuit netlists by orders of magnitude and will be a focus of future work. Table I lists all the geometric and LDE transistors parameters to be predicted.

The number of fanout of each net as well as various device sizing features are used as inputs to the model. Table II lists all the features extracted for each device type and net.

B. Graph Construction

To construct a graph based on a circuit schematic, each device (transistor, resistor, capacitor, etc.) can be mapped into nodes within the graph. To predict net capacitance, we choose to map nets as nodes in the graph as well. This ensures nets are represented as graph elements with corresponding node information (i.e. net capacitance) that would otherwise be lost if nets were represented as graph edges. To complete the structure, two edges with opposing directions are mapped between every net node and the appropriate device nodes corresponding to terminal connections within the schematic.

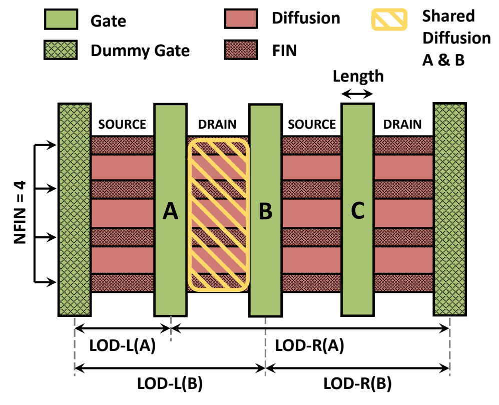


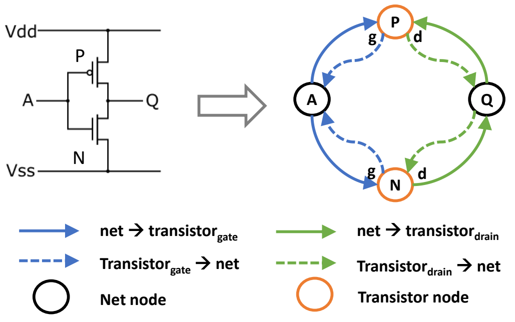
Fig. 2: FinFET Transistor Layout Illustration

Type	Parameters	Definition
transistor	LDEx	x=(1,..,8), eight LDE parameters
	SA,DA	source and drain diffusion areas
	SP,DP	source and drain diffusion perimeters
net	CAP	net parasitic capacitance

TABLE I: List of Transistor Parameters and Net Parasitics

Type	Features	Definition
transistor & transistor$_{thickgate}$	L	gate poly length
	NF	number of fingers
	NFIN	number of fins
	MULTI	number of copies (multiplier)
resistor	L	length of resistor
capacitor	MULTI	number of copies
dio	NF	number of fingers
bjt	1	constant
net	N	number of fanout

TABLE II: List of Devices and Net Features



Vdd
P
A
Q
N
Vss
P
g
d
A
Q
g
N
d
net → transistor$_{gate}$
Transistor$_{gate}$ → net
Net node
net → transistor$_{drain}$
Transistor$_{drain}$ → net
Transistor node

Fig. 3: Inverter Schematic and Heterogeneous Graph

The benefit of mapping both net and devices into graph nodes is that the relationship between devices and nets is explicit within the graph and can therefore be used in learning. This naturally supports hyperedge connections between devices since each net node can connect to multiple device nodes. Additionally, multiple edge relationships can be supported between net nodes and device nodes based on device terminal types (i.e. source, gate, drain). The ability to make distinctions between device terminals will help the learning model differentiate different connection structures. Connections to supply and ground nets are ignored within the graph since predicting parasitics on the power-rails is not necessary, which in turn reduces overall graph complexity.

The resulting graph is a heterogeneous Graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ consisting of a node set $\mathbb{V}$ and a directed edge set $\mathbb{E}$. The heterogeneous graph is also associated with a node type mapping function $\phi : \mathbb{V} \rightarrow \mathbb{T}_{\mathbb{N}}$ and an edge type mapping function $\psi : \mathbb{E} \rightarrow \mathbb{T}_{\mathbb{E}}$, where $\mathbb{T}_{\mathbb{N}}$ and $\mathbb{T}_{\mathbb{E}}$ are sets of node types $\{transistor, net, capacitor, resistor, etc\}$ and edge types $\{net \rightarrow transistor_{gate}, transistor_{gate} \rightarrow net, net \rightarrow transistor_{source}, etc\}$, respectively. In the heterogeneous graph there are always two edges with opposing directions between two nodes, which correspond to differing edge types. For example, the opposite edge of type $net \rightarrow transistor_{gate}$ is an edge of type $transistor_{gate} \rightarrow net$. Figure 3 shows the heterogeneous graph of an inverter.

III. Graph Neural Networks

Graph neural networks (GNN) are deep learning models for the graph domain [8] and are based on two core ideas: graph embedding and neighbor aggregation. Graph embedding learns embedded features by transforming graph nodes, edges, subgraphs, and their corresponding features into a lower-dimensional vector space representation. Neighbor aggregation learns the embedding properties of a node by aggregating information from nodes within its local neighborhood, similar to convolution on neighboring image pixels in Convolutional Neural Networks (CNN). Compared to direct embedding methods such as Deep-walk [9] and node2vec [10], which map nodes to vector embedding by a simple embedding lookup, GNN-based methods can be generalized to graphs never seen by the network since graph structures are directly incorporated into the learning algorithm.

There are four widely used GNN models: Graph Convolutional Network (GCN) [3], GraphSage [4], Relational GCN (RGCN) [5] and Graph Attention Network (GAT) [6]. These models differ in how neighboring information is aggregated. Table III shows the comparison of these GNN models.

In Table III, $h_i^{(l)} \in \mathbb{R}^F$ is the embedding for the lth layer with F dimensions, $h_i^{(l+1)}$ is the updated embedding for layer $l+1$, $N(i)$ is the neighboring set for node i, c_{ij} is equal to the product of the square root of node degrees such that $c_{ij} = \sqrt{|N(i)|}\sqrt{|N(j)|}$, σ is an activation, $norm$ is a normalization function function, $b^{(l)}$ is the bias of the lth layer, $N^r(i)$ is the neighboring set of node i w.r.t. relation r, $c_{i,r}$ is the normalizer equal to $|N^r(i)|$, W_0 is the self-loop weight, W_r is the weight for relation r, $\alpha_{i,j}$ is the learned attention between node i and j, and $\vec{a}$ is the attention matrix: $\mathbb{R}^F \times \mathbb{R}^F \rightarrow \mathbb{R}$.

GCN and GraphSage are two early GNN models that both leverage a 'convolutional' mean aggregator to approximate localized spectral filters. GraphSage differs from GCN by performing a concatenation of previous embeddings with the aggregated neighbor embeddings. This concatenation is similar to the “skip connection” [11] between different layers of the GraphSAGE algorithm. For graphs with multiple relational edges, RGCN is proposed for distinguishing between different relational edges. It applies different weight matrices to different relational edge groups and aggregates each group independently. GAT introduces a self-attention mechanism commonly used in sequence models to replace the mean neighborhood aggregator. It allows different weights to be associated with each neighbor in order to increase the modeling capacity. Analyzing the learned attentional weights may also help model interpretability.

Model	**Aggregation Formula**	
GCN	$h_i^{(l+1)}$	$= \sigma(b^{(l)} + \sum_{j \in N(i)} \frac{1}{c_{ij}} W^{(l)} h_j^{(l)})$
GraphSage	$h_{N(i)}^{(l+1)}$	$= mean\left(\{h_j^{(l)}, \forall j \in N(i)\}\right)$
	$h_i^{(l+1)}$	$= \sigma\left(W \cdot concat(h_i^{(l)}, h_{N(i)}^{(l+1)} + b^{(l)})\right)$
	$h_i^{(l+1)}$	$= h_i^{(l+1)} / \|h_i^{(l+1)}\|_2)$
RGCN	$h_{N(i)}^{(l+1)}$	$= \sum_{r \in R} \sum_{j \in N^r(i)} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)}$
	$h_i^{(l+1)}$	$= \sigma(h_{N(i)}^{(l+1)} + W_0^{(l)} h_i^{(l)})$
GAT	e_{ij}^l	$= \vec{a}^T concat(W^{(l)} h_i^l, W^{(l)} h_j^l)$
	α_{ij}^l	$= softmax_i\left(LeakyReLU(e_{ij}^l)\right)$
	$h_i^{(l+1)}$	$= \sigma\left(\sum_{j \in N(i)} \alpha_{i,j} W^{(l)} h_j^{(l)}\right)$

TABLE III: Graph Neural Networks Comparison

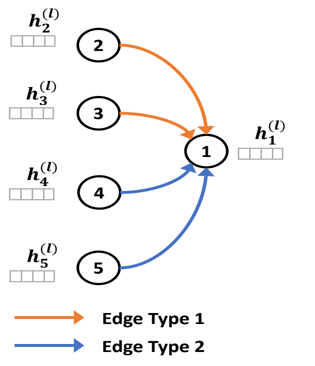
All of the GNN models listed in Table III except RGCN assume the graph is homogeneous, meaning there is only one node type and edge type in the graph. Even though RGCN can model different edge types, it is still limited to a single type of node. This is problematic since graphs converted from circuit schematics are heterogenous. They contain multiple node types that correspond to nets and various types of devices ($transistor$, $resistor$, $capacitor$, etc) where each node type has unique feature dimensions as shown in Table II. Furthermore, different edge relationships exist between differing node types ($net \rightarrow transistor_{gate}$, $net \rightarrow transistor_{source}$, etc). This results in graph networks that contain more information than the homogenous graphs typically learned by a GNN, therefore requiring the development of a graph model capable of handling circuit graphs. There are no well-known heterogeneous GNN models, and the problem domain of circuit graphs is unique. Therefore we developed our own model.

We call our GNN model ParaGraph. It includes a node embedding model and several fully connected (FC) layers, which take node embedding as inputs and generates predictions for a layout parasitics or a device parameter. We train independent models for predicting each device parameter as well as layout parasitics.

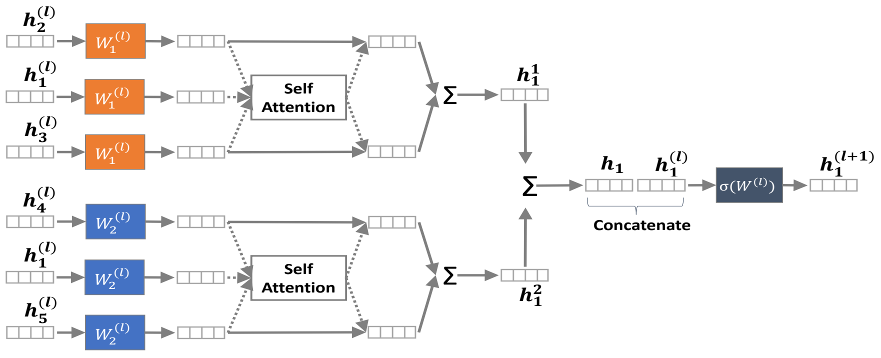
Algorithm 1 Node Embedding Algorithm in ParaGraph

```
Require: Graph $\mathbb{G}(\mathbb{V}, \mathbb{E})$, node types $\mathbb{T}_{\mathbb{N}}$, edge types $\mathbb{T}_{\mathbb{E}}$, node type
    mapping $\phi : \mathbb{V} \rightarrow \mathbb{T}_{\mathbb{N}}$, edge type mapping $\psi : \mathbb{E} \rightarrow \mathbb{T}_{\mathbb{E}}$, node
    features $\{h_i, \forall i \in \mathbb{V}\}$, layer depth $L$
Output: final Node embedding $\mathbb{Z} = \{z_i, \forall i \in \mathbb{V}\}$
 1: for node type $t \in N_T$ do
 2:     $h_i^{(0)} \leftarrow W_t h_i, \forall \phi(i) = t$        ▷ map to common feature space
 3: for $l \leftarrow 0$ to $L-1$ do                                 ▷ layers
 4:     for $t \in E_T$ do                                          ▷ edge types
 5:         for $edge(i, j)$ with $\psi(edge) = t$ do               ▷ attentions
 6:             $e_{ij} \leftarrow \vec{a}^T concat\left(W_t^{(l)} h_i^{(l)}, W_t^{(l)} h_j^{(l)}\right)$
 7:             $\alpha_{ij} \leftarrow softmax_i\left(LeakyReLU(e_{ij})\right)$
 8:         $h_i^t \leftarrow \sum_{j \in N_t(i)} \alpha_{i,j} W_t^{(l)} h_j^{(l)}, \forall i \in \mathbb{N}$
 9:     $h_i \leftarrow \sum_{t \in E_T} h_i^t, \forall i \in \mathbb{N}$
10:     $h_i^{(l+1)} \leftarrow \sigma\left(W^{(l)} \cdot concat(h_i^{(l)}, h_i + b^{(l)})\right)$
11: $\mathbb{Z} \leftarrow \{z_i = h_i^{(l)}, \forall i \in \mathbb{V}\}$
```

To compute node embedding, we leverage the key ideas of GraphSage, RGCN and GAT. First, similar to GraphSage, we use the



(a) Node 1 has four input edges with two different edge types. Each node has its own embedding on layer l.



(b) Compute graph of a ParaGraph embedding layer. Each input node embedding is fed into different weight metrics based on their input edge types, aggregated with self attention, then concatenated with target node's layer l embedding before fed into a weight matrix to generate embedding on layer $l + 1$.

Fig. 4: Example of the ParaGraph Node Embedding Update.

concatenation of previous embeddings with the aggregated neighbor embeddings. Second, we group different edge types independently during aggregation similar to RGCN. Lastly, we add a self-attention layer in between the aggregation of each group similar to GAT. The node embedding algorithm of ParaGraph is outlined in Algorithm 1 and an example of the compute graph for each embedding layer is illustrated in Figure 4.

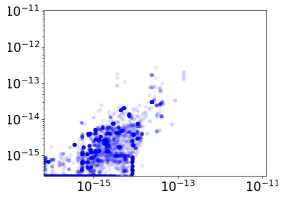
To predict a specific parasitic y on node type t, we feed node embedding z_i to several FC layers where $\phi(i) = t$. All of the FC layers except the last one have the same dimension F as the embeddings, while the last layer has only 1 dimension. We use a Mean Square Error (MSE) loss function to regress the predicted value against the ground truth value.

IV. Ensemble Modeling

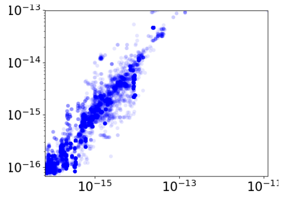
In our analog and mixed-signal circuit dataset, net parasitic capacitances range from 0.01fF to 10pF, resulting in prediction values that span more than 6 orders of magnitude. This makes training a single model for predicting across the entire range difficult since the inherent model error due to layout uncertainty is already much larger than 1%. This means any capacitance value less than 1% of the maximum predicted value (< 0.1pF) will be considered noise by the model and therefore results in poor prediction accuracy for small capacitance values.

Figure 5a shows the predicted net capacitance vs ground truth for all testing circuits predicted with a model trained on the maximum capacitance range (0.01fF - 10pF). The predictions become increasingly inaccurate (i.e. deviates from a diagonal line) for ground truth values smaller than 100fF. To alleviate this problem, we propose training multiple models each with different maximum prediction values (max_v = 100fF, 10fF, and 1fF). We call these models 100fF model, 10fF model, and 1fF model, respectively. Data points with a ground truth larger than the maximum predicted value are ignored during training, which increases prediction accuracy within a specified range within each model (i.e. values within two orders of magnitude of the max). The predicted and ground truth parasitics capacitances inferenced with these three models are shown in Figure 5b, 5c and 5d, respectively.

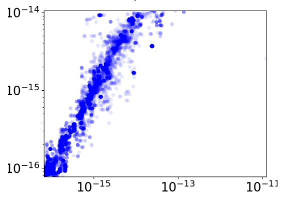
To leverage ensemble modeling for better prediction accuracy, we choose which model to use in predicting individual net capacitances. Models with higher maximum prediction values are more accurate in the higher range than the models with lower maximum prediction values, e.g. the 100fF model is more accurate in the range of



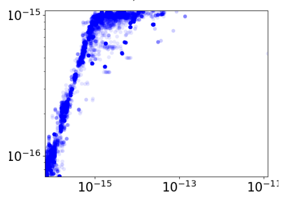
(a) max_v=10pF
MAE=1.44fF, MAPE=133%



(b) max_v=100fF
MAE=1.47fF, MAPE=31.6%



(c) max_v=10fF
MAE=1.40fF, MAPE=18.6%



(d) max_v=1fF
MAE=2.02fF, MAPE=23.0%

Fig. 5: Parasitics capacitance prediction with models trained with different max_v. (x-axis is the original, y-axis is the prediction.)

Algorithm 2 Ensemble Modeling For Net Parasitics Capacitance

```
Require: a list of $K$ models $M_i$, and max prediction value $max_v^i$,
    $i \in (1, .., K)$, $max_v^0 < max_v^1 < max_v^2 ... < max_v^K$, and net $n$
Output: prediction $p(n)$ for net $n$
1: $p(n) \leftarrow$ predicting net $n$ with $M_1$
2: for $i \leftarrow 2$ to $K$ do
3:     $p'(n) \leftarrow$ predicting net $n$ with $M_i$
4:     if $p'(n) > max_v^{i-1}$ then
5:         $p(n) \leftarrow p'(n)$
```

[10fF,100fF] than the 10fF model in that range. The algorithm to select prediction values based on different models is given in Algorithm 2. Based on this algorithm, if the 10fF model predicts a value of 2.5fF, which is greater than the maximum predicted value of 1fF model, it will be preferred than the 1fF model.

Figure 7a shows predicted capacitance from the ensemble model vs ground truth. Through visual inspection, we can see that the predictions with the ensemble model are more accurate over the entire prediction range. Quantitatively, the ensemble model gives the smallest mean absolute error (MAE): 0.852fF and mean absolute percentage error (MAPE): 15.0% of all the individual models.

V. Experimental Results

The models were evaluated on a dataset of industrial analog and mixed-signal circuits built within a sub-10nm technology. The circuits contain various active and passive devices including MOSFETs, BJTs, diodes, resistors and capacitors. The training and testing data set split was based on designer recommendation, which ensures the test set circuits are completely different than those in the training set. The characteristics of the dataset are shown in Table IV.

circuit	#net	#tran	#tran$_{th}$	res	cap	bjt	dio
t1	94	176	0	0	0	0	0
t2	5027	3306	4981	15	250	0	0
t3	15314	5031	18192	20	1503	0	14
t4	196957	327856	88209	135	2758	0	0
t5	93706	146655	4250	293	351	0	0
t6	91792	142575	4250	6	351	0	0
t7	62227	64890	2641	3	330	54	0
t8	2083	0	2843	2	0	0	0
t9	2103	0	2899	2	0	0	0
t10	61666	114822	0	0	0	0	0
t11	35820	689	48303	21	330	36	0
t12	4400	8486	0	0	0	0	0
t13	32643	57377	0	0	0	0	0
t14	1203	44	1477	0	112	0	6
t15	62124	48298	42221	12	566	36	0
t16	15408	37320	0	0	0	0	0
t17	37310	6262	39906	18	548	153	0
t18	5570	11077	1	0	90	0	2
e1	11271	22461	0	0	0	0	0
e2	707	188	1003	0	8	0	8
e3	3998	9193	0	0	0	0	0
e4	4672	11674	0	0	0	0	0

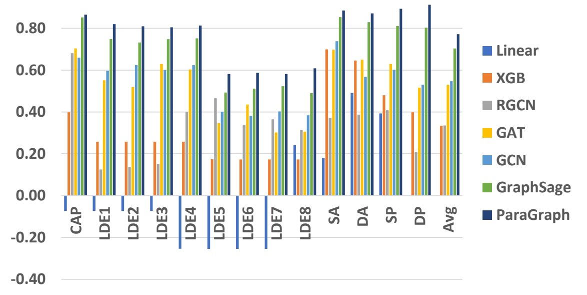
TABLE IV: Device and Net Distribution of the Circuit Dataset. $t1-t18$ are used in training while $e1-e4$ are used for testing.

The graph models were implemented with Deep Graph Library (DGL) [12], which is based on PyTorch [13], since DGL provides many convenient APIs for building graph convolution layers. The proposed ParaGraph model as well as naive GCN, GraphSage, RGCN and GAT networks were implemented for characterization on the dataset. Node type specific transformations (Algorithm, line 1-2) had to be applied for the naive GCN, GraphSage, RGCN and GAT models to get a common feature space for each node. All the models are trained on a single NVIDIA Tesla V100 GPU with 16GB memory.

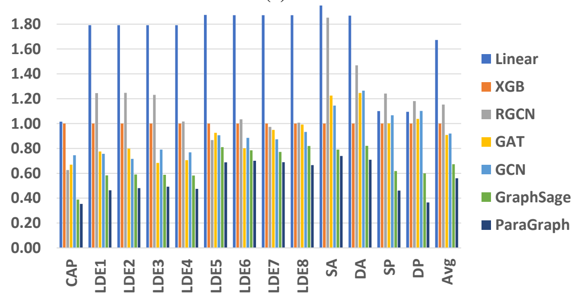
For each graph model, the dimension of embedding and number of layers were set to $F = 32$ and $L = 5$, respectively. We swept the number of layers and found a higher number of layers gives better results and plateaus at 5. Both GAT and ParaGraph models can potentially use more than one attention head, however we are limited by GPU memory to only use one attention head on our dataset. We expect more attention heads would lead to even better results. The number of FC layers were set to 4 for the net parasitic capacitance model and 2 for all device parameter models. The ADAM optimizer with a learning rate of 0.01 was used for training. Since the training sets are sufficiently large, there was no need to apply L2 regularization or dropout. We train each graph model for 300 epochs and measure the average prediction accuracy across 10 runs.

To compare the accuracy of different graph models, we use three statistical measurements: R-squared (R^2), Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE). We also include two classical ML models: XGBoost [14] and Linear Regression based on node features alone in the comparison. A single net parasitic capacitance model max_v=10fF is used in this study to ensure the model comparison is not biased by the ensemble modeling.

Figure 6 shows the average prediction R^2 and MAE (relative to the XGB model) on all testing circuits over the 10 runs. Similar



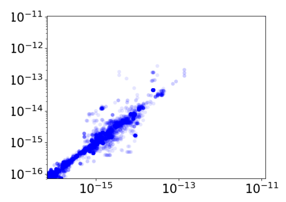
(a) R^2



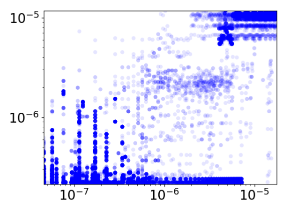
(b) MAE relative to XGBoost model

Fig. 6: Prediction accuracy comparison for different learning models

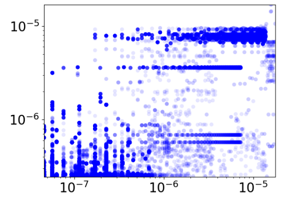
comparison is observed for MAPE, and is omitted for brevity. ParaGraph produces the best R^2 value for all device parameters and net parasitic capacitances with an average prediction R^2 of 0.772 (a perfect prediction would have a maximum R^2 value of 1), which is 110% better than XGBoost. The second best model is GraphSage with an average prediction R^2 of 0.703. Furthermore, ParaGraph has the smallest MAE and MAPE compared to all other learning models. It reduces the MAE and MAPE of XGBoost by 44% and 67.3%, respectively, while the second best model GraphSage only by 33% and 58.3%, respectively.



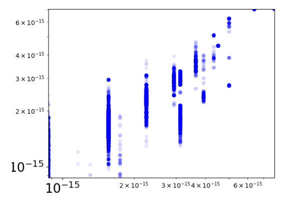
(a) CAP (Ensemble Model)
MAE=0.852fF, MAPE=15.0%



(b) LDE1
MAE=0.921um, MAPE=162%



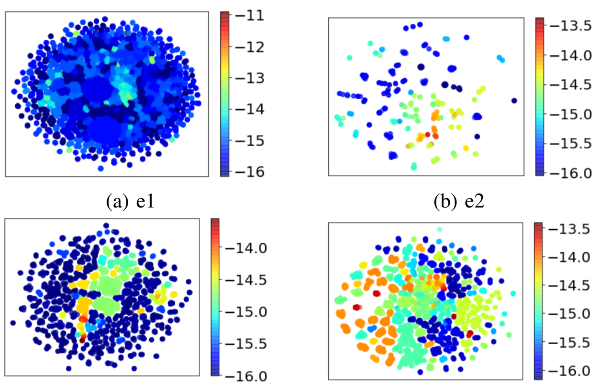
(c) LDE5
MAE=1.00um, MAPE=241%



(d) SA
MAE=193nm^2, MAPE=10.3%

Fig. 7: Device parameters (LDE1, LDE5, SA) and net parasitics (Capacitance) prediction vs ground truth on all testing circuits. (x-axis is the original values, y-axis is the predicted values.)

Figure 7 shows the comparison of the ParaGraph predictions and ground truths on the net parasitic capacitance, two of the LDE parameters, and the device source area (SA) parameter. The net



(a) e1 (b) e2

(c) e3 (d) e4

Fig. 8: t-SNE plots of embeddings of capacitance model max_v=10fF on each testing circuit. Colored by log10 of the ground truth.

Error Range	Layout w/o parasitics	Designer's Estimation	Prediction w/ XGB	Prediction w/ ParaGraph
$< 10\%$	4	6	17	44
10%-20%	0	17	14	10
20%-30%	5	18	4	8
30%-40%	35	2	7	4
40%-50%	14	6	9	1
$> 50\%$	9	18	16	0
Mean	37.75%	>100%	32.14%	9.60%
Geometric Mean	29.01%	43.57%	15.46%	4.00%

TABLE V: Simulation errors between pre-layout predictions and post-layout on 67 circuit metrics in the testing circuits.

parasitic capacitance and the device source area parameter have a very good match between the predictions and ground truths, while the LDE parameters have less accurate predictions. The MAPEs are 15.0% and 10.3% for parasitics capacitances and device SA parameters, respectively; while the MAPEs for both LDE parameters are more than 100%. We believe the relatively large prediction errors on LDE parameters are the result of inherent layout uncertainty.

To investigate the meaning of the embeddings, we can visualize the high dimensional embedding vector with t-SNE algorithm [15]. Figure 8 shows t-SNE plots for net node embeddings of the capacitance model trained with max_v=10fF on all testing circuits. The nodes are colored according to the log10 of their ground truth values. Data points with different colors are well separated, indicating the model learned to differentiate nets with different capacitances. Note that there are minor overlapping colors for data points greater than 10fF. This is expected since the model is trained with max_v=10fF. Nets with capacitance higher than 10fF will result in inaccurate predictions, which explains why ensemble modeling is needed.

To evaluate the effectiveness of the prediction model, pre-layout simulations are run using the predicted parasitics and device parameters predicted by both the XGBoost and ParaGraph models. The errors of a total of 67 key circuit metrics (i.e. slew rate, insertion delay, power, etc) within the testing circuits are calculated in comparison to simulations using the post-layout netlist. Comparisons are also made to two baseline simulations: one with parasitic annotations from the designer's original estimates, and the other using layout netlist without extracted parasitics. Layout parasitics capacitance estimation flags in the spice model are disabled for simulations with predicted parasitics, but enabled for two baseline simulations. The resulting simulation errors for each testing circuit are shown in Table V. On average, the ParaGraph prediction reduces the pre-layout simulation error geometric mean from 43.57% and 29.01% in two baselines to only 4.00%. The mean error is also reduced to 9.60%. The number of metrics with relative large errors ($> 30\%$) is significantly reduced. The designer's estimation manages to reduce simulation errors on some metrics, but significantly increases errors on others metrics that are highly sensitive to parasitics, which results in larger average errors. Note that even with ParaGraph model, there are still several simulation error $> 30\%$. This is because these circuit metrics depend on nets with larger parasitics capacitance, e.g. greater than 10fF. The predictions errors are generally worse for those larger parasitics for a couple of reasons: first, circuit floorplan has large impact on larger parasitics, which is hard to predict from schematic; and second, our training data is limited for larger parasitics, we would expect accuracy to increase with more data.

VI. Conclusion

In this paper, we apply machine learning to predict layout parasitics and device parameters to increase the accuracy of pre-layout simulations of analog and mixed-signal circuits. Inspired by the observation that similar circuit structures produce similar parasitics, we built a machine learning model (ParaGraph) based on key ideas of state-of-art graph neural networks. The model is evaluated on a large dataset of industrial circuits based on electrical simulations using the predictions. Based on the prediction accuracy and reduction in simulation errors, we are optimistic that graph convolutional networks can be used in this task. Future work will focus on extending this model to predict net parasitic resistances as well.

VII. Acknowledgements

We wish to thank the MAGESTIC team from Cadence Design System, Inc. for helpful discussions.

References

[1] L. Zhang, N. Jangkrajarng, S. Bhattacharya, and C.-J. R. Shi, "Parasitic-aware optimization and retargeting of analog layouts: A symbolic-template approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, pp. 791–802, 2008.
[2] H. Yoshida, K. De, and V. Boppana, "Accurate pre-layout estimation of standard cell characteristics," in *DAC*, 2004, pp. 208–211.
[3] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016.
[4] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1025–1035.
[5] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," *Lecture Notes in Computer Science*, p. 593–607, 2018.
[6] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *ArXiv*, vol. abs/1710.10903, 2017.
[7] Y. Ma *et al.*, "High performance graph convolutional networks with applications in testability analysis," in *DAC*, 2019, pp. 18:1–18:6.
[8] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *CoRR*, vol. abs/1812.08434, 2018.
[9] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," *CoRR*, vol. abs/1403.6652, 2014.
[10] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.
[11] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *Lecture Notes in Computer Science*, p. 630–645, 2016.
[12] M. Wang *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
[13] A. Paszke *et al.*, "Automatic differentiation in pytorch," in *NeurIPS Autodiff Workshop*, 2017.
[14] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of KDD*, 2016, pp. 785–794.
[15] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, pp. 2579–2605.