Parasitic-Aware Analog Circuit Sizing with Graph Neural Networks and Bayesian Optimization

Mingjie Liu[1,⋆], Walker J. Turner[2], George F. Kokai[2], Brucek Khailany[2], David Z. Pan[1], and Haoxing Ren[2,†]

[1]ECE Department, The University of Texas at Austin [2]NVIDIA
⋆jay_liu@utexas.edu †haoxingr@nvidia.com

***Abstract*—Layout parasitics significantly impact the performance of analog integrated circuits, leading to discrepancies between schematic and post-layout performance and requiring several iterations to achieve design convergence. Prior work has accounted for parasitic effects during the initial design phase but relies on automated layout generation for estimating parasitics. In this work, we leverage recent developments in parasitic prediction using graph neural networks to eliminate the need for in-the-loop layout generation. We propose an improved surrogate performance model using parasitic graph embeddings from the pre-trained parasitic prediction network. We further leverage dropout as an efficient prediction of uncertainty for Bayesian optimization to automate transistor sizing. Experimental results demonstrate the proposed surrogate model has 20% better R^2 prediction score and improves optimization convergence by 3.7 times and 2.1 times compared to conventional Gaussian process regression and neural network based Bayesian linear regression, respectively. Furthermore, the inclusion of parasitic prediction in the optimization loop could guarantee satisfaction of all design constraints, while schematic-only optimization fail numerous constraints if verified with parasitic estimations.**

I. Introduction

Analog design, and especially analog layout, is a labor-intensive process still lacking high-quality design automation tools. Some difficulties include the larger degree of freedom compared to digital design and the design-specific performance metrics of analog circuits. As a result, layout is still usually performed manually and design often involves iterative optimization between schematic and layout. To avoid layout surprises, designers often estimate crucial parasitic effects during the initial schematic design phase. However, the inability to accurately estimate layout parasitics often leads to costly iterations between the electrical and physical designs and limits the ability to do schematic-level design space exploration such as automated transistor sizing. This issue is exacerbated in deep sub-micron technologies as parasitic are becoming increasingly difficult to model with simple heuristics.

Numerous methods have previously been proposed for automated transistor sizing. Early work [1] generated symbolic AC models and optimized circuit performance using simulated annealing. Equation-based methods have also been proposed [2]. Recent work focused on directly using results from high fidelity performance simulations and leveraged black-box gradient-free optimization methods. Model-free algorithms such as differential evolution [3] are typically less sample efficient than model-based methods. Bayesian optimization [4], [5] and reinforcement learning [6] are examples of model-based methods where either Gaussian Process regression or neural network models are trained concurrently with optimization. However, designers are reluctant to adopt schematic-only automatic sizing methodologies, since layout-dependent parasitic effects are usually disregarded during pre-layout simulations.

To achieve design closure, recent work has considered layout-dependent parasitic effects for automated analog sizing. The approaches either fully embed automated layout generators with parasitic extraction inside the sizing optimization loop or estimate the impact of parasitics after layout placement. Ranjan et al. [7] proposed fast performance estimation based on design sampling from parameterized layout generators. Habal et al. [8] enumerated device layouts and optimized sizing using nonlinear optimization. The work of [9] estimated parasitics after placement prior to detailed routing. BagNet [10] and AutoCkt [11] both leverage Berkeley Analog Generator [12] to generate layouts, and optimize sizing with evolutionary algorithm and reinforcement learning, respectively. Although automatic analog layout generation is an active field of research, mature commercial tools still have not been widely adopted by designers, and these methods are typically only used in research settings.

Recent advancements in machine learning have enabled statistical and data-driven approaches to accurately estimate layout parasitics directly from circuit schematics. ParaGraph [13] proposes the use of a graph neural network (GNN) model to predict net parasitic capacitance by converting circuit schematics into heterogeneous graphs. MLParest [14] trains Random Forest models based on extracted features from circuit schematic netlists to predict an effective resistance and lumped parasitic capacitance. Both approaches have demonstrated reduced errors when comparing pre-layout and post-layout circuit simulation results. Graph neural networks have also received increasing attention in assisting analog circuit design, such as transistor sizing [6], layout constraint extraction [15], [16], and guiding analog placement [17].

In this paper, we present a parasitic-aware sizing methodology leveraging state-of-the-art parasitic prediction models. Compared with prior work, our method does not rely on automatic layout generation. We further improve the surrogate model with a parasitic graph embedding enhanced neural network. Our main contributions are summarized as follows:

- We present a parasitic-aware sizing methodology based on machine learning parasitic prediction, replacing in-the-loop layout generation.
- We extend prior GNN-based parasitic prediction work to include parasitic resistance and coupling capacitance, reducing simulation errors by 41.3%.
- We propose a surrogate model that utilizes graph embeddings from the pre-trained GNN parasitic prediction

model, improving R^2 prediction score by 20%.

- We use dropout in neural networks as an efficient method of predicting uncertainty for Bayesian optimization.
- Experimental results demonstrate that the proposed method increase optimization convergence by 2.1 times compared with prior state-of-the-art approach [18].

The rest of the paper is organized as follows. Section II gives the background of circuit sizing and Bayesian optimization; Section III presents parasitic resistance and coupling capacitance prediction with graph neural networks; Section IV explains the details of the parasitic-aware circuit sizing framework, improved surrogate modeling, and uncertainty prediction leveraging dropout; Section V demonstrates the experimental results; Section VI concludes the paper.

II. Background and Preliminaries

In this section we first formulate the circuit sizing problem in Sec. II-A. Then we give a brief overview of the two key ingredients of Bayesian optimization: probabilistic modeling in Sec. II-B and acquisition functions in II-C.

A. Problem Formulation

Analog circuit optimization typically involves several jointly optimized performance metrics. Most work simplifies the problem by designing a single figure of merit (FOM) objective with a weighted average of different performance metrics. The weights need to be predetermined by the designer, which can lead to a bias of prioritizing certain performance metrics over others.

In practice, designers often aim at providing functional coverage and thus optimize circuits to satisfy performance constraints while minimizing the cost in power or layout area. We thus formulate analog sizing into a constrained single objective optimization problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0(x) \\ \text{subject to} \quad & c_i(x) \le b_i,\ i = 1, \ldots, m, \end{aligned} \tag{1}$$

where x is the d-dimensional design variable, $f_0(x)$ is the selected cost to minimize (typically power), and $c_i(x)$ are m performance metrics to meet the target of b_i.

B. Bayesian Linear Regression and Gaussian Process

Bayesian linear regression (BLR) is a probabilistic model for linear regression tasks, in the form of $y = x^T w + b$, where y is the surrogate function output, and $b \sim N(0, \sigma_n^2 I)$ is the observed noise. Given a prior distribution on weights $w \sim N(0, \sigma_w^2 I)$, the predictive mean and variance are as follows:

$$\mu(x^*) = \sigma_n^{-2} x^{*T} K^{-1} \Phi_D, \tag{2}$$

$$\sigma(x^*)^2 = x^{*T} K^{-1} x^* + \sigma_n^2, \tag{3}$$

$$K = \sigma_n^{-2} \Phi_D \Phi_D^T + \sigma_w^{-2} I, \tag{4}$$

where Φ_D is the design matrix of training data $D \in \{X, y\}$.

Gaussian process regression (GPR) is a non-parametric surrogate model, which places Gaussian priors over the latent function with a mean function $m(x)$ and covariance kernel function $k(x, x')$. The constant mean function $m(x) = \mu_0$ is commonly used. An example of the kernel function is the Radial Basis Function with additive white noise:

$$k(x, x') = \sigma_f^2 \exp(-\frac{1}{2}(x - x')^T \Lambda^{-1}(x - x')) + \sigma_n^2 I, \tag{5}$$

where hyperparameters σ_n^2, σ_f^2 and l_j are the observation noise variance, signal variance and length-scales along the jth dimension. Given a new data point x^*, the prediction mean and variance are derived as follows:

$$\mu(x^*) = \mu_0 + k(x^*, X) K_\theta^{-1}(y - \mu_0), \tag{6}$$

$$\sigma(x^*)^2 = k(x^*, x^*) - k(x^*, X) K_\theta^{-1} k(X, x^*). \tag{7}$$

Recent work [19] has proposed replacing the last layer of neural networks with Bayesian linear regression as a scalable approach for Gaussian process regression. Zhang et al. [18] were the first to adopt similar ideas for analog circuit sizing using neural networks as trainable feature maps to substitute kernel functions in GPR models.

C. Acquisition Function

Bayesian optimization leverages probabilistic models to predict uncertainty with an acquisition function to balance exploration and exploitation. To handle the constrained optimization in Sec. II-A, we use the weighted Expected Improvement (wEI) [4]. Each objective and constraint is modeled individually. The Expected Improvement (EI) of the objective f_0 over the incumbent solution f^* with Φ given as the CDF of the normal distribution is as follows:

$$\begin{aligned} EI(x) &= E_{f_0 \sim N(\mu_0, \sigma_0)}[max(f^* - f_0, 0)] \\ &= (f^* - \mu_0)\Phi(\frac{f^* - \mu_0}{\sigma_0}) + \sigma_0 \Phi(\frac{f^* - \mu_0}{\sigma_0}). \end{aligned} \tag{8}$$

The probability of feasibility (PoF) for satisfying all constraints is expressed as:

$$PoF(x) = \prod_{i=1}^{m} Pr(c_i(x) < 0) = \prod_{i=1}^{m} \Phi(\frac{-\mu_i}{\sigma_i}). \tag{9}$$

Weighted Expected Improvement is the product of EI and PoF where the EI is calculated accordingly to (8) if there exists an incumbent feasible solution, or disregarded and treated as constant if all previous candidates are infeasible.

III. Parasitic Prediction with Graph Neural Networks

ParaGraph [13] proposes a GNN model to predict net parasitics and device parameters by converting circuit schematics into graphs with heterogeneous edge and node types. Each device and net is mapped into nodes within the graph. Nodes are connected in the graph based on the circuit topology, while different edges are determined by the connected pin type between the net and device as shown in Fig. 1(a). To compute the next layer of the node embeddings, the previous layer node embeddings of the target node neighbors are fed into different weight matrices based on the edge type, aggregated with self-attention, concatenated with the target nodes embedding and fed into a final linear layer to generate the target's next layer embedding. To predict a parasitic value, the target node embedding is fed into a neural network with fully connected layers for regression in Fig. 1(b).

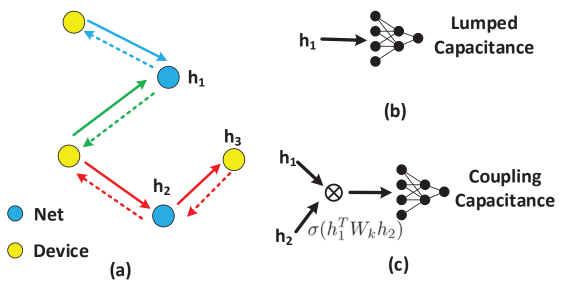


Fig. 1: Improved Parasitic Prediction. (a) ParaGraph node embedding with heterogeneous graphs (b) Lumped capacitance prediction (c) Coupling prediction with bilinear layer

In this work, we extend Paragraph's capabilities by adding predictions to the model for parasitic resistance and coupling capacitance. As shown in Fig. 2, layout extraction creates a distributed RC network with multiple intermediate nodes for a single net in the schematic. We only model the effective resistance of each net to the connected transistor gates where the effective resistance is obtained with DC simulations as show in Fig. 2(c). The final net parasitic model is shown in Fig. 2(d) with only the effective resistance of gate connections, lumped capacitance and coupling capacitance. Compared with [14], where the resistance is uniform and coupling is disregarded, the extended ParaGraph parasitic model has coupling capacitance and non-uniform effective resistance modeled.

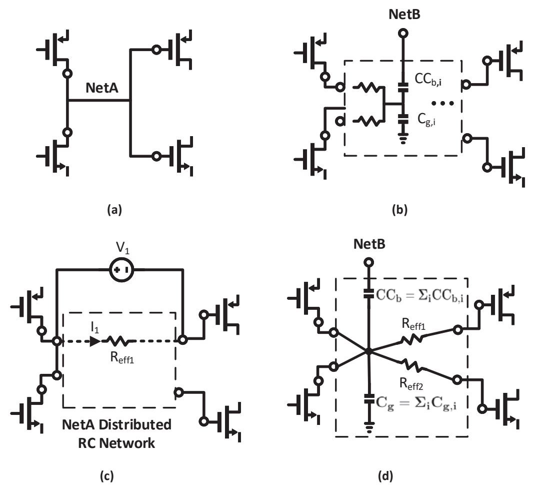


Fig. 2: Parasitic Modeling. (a) Circuit schematic (b) Distributed RC network from extraction (c) Measurement of effective resistance for labeling (d) Proposed parasitic model

Resistance and coupling regression tasks are formulated as similar tasks to link prediction in social network analysis [20]. Where net lumped capacitance only involves a single graph node, coupling and resistance need information from both source and target nodes. The entire prediction process is mapped in two stages, (1) the node embeddings are obtained by aggregating local neighborhood graph features with graph neural networks, and (2) the final regression network where both the source and target node embeddings are the input. The regression network demonstrated in Fig. 1(c) consists of a bilinear layer followed with several layers of fully connected networks. The bilinear layer is more suitable for modeling pairwise feature interactions, calculated as follow:

$$f(h_1, h_2) = h_1^T W_k h_2, \tag{10}$$

where h_1, h_2 are the input node embeddings of the source and target, and W_k are k trainable weight matrices to generate an output vector of size k.

IV. Parasitic-Aware Analog Sizing

In this section we present the details of the parasitic-aware circuit sizing. We provide an overview of the framework in Sec. IV-A. We present the improved performance modeling with parasitic graph embedding in Sec. IV-B and the method of leveraging dropout to model uncertainty in Sec. IV-C.

A. Automated Analog Sizing with Parasitic Predictions

The parasitic-aware sizing framework is depicted in Fig. 3. During each optimization iteration, the design parameters are updated with Bayesian optimization. The parasitic prediction engine (ParaGraph) predicts post-layout parasitics for the circuit based on the circuit topology and design parameters, which are back-annotated into the schematic design. The circuit FOM is obtained with SPICE simulations and added to the training set for the surrogate model. The neural network models the objectives and constraints and is trained every iteration with information from both the design variables and parasitic information in ParaGraph. Finally the acquisition function is maximized using the surrogate model to obtain the next design parameters.

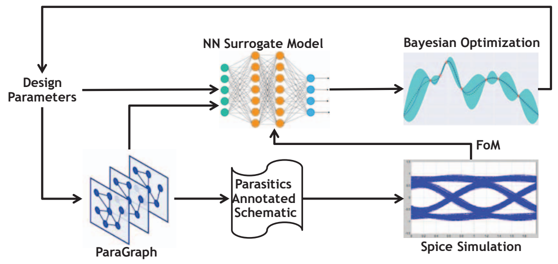


Fig. 3: Parasitic-aware analog sizing framework.

The key differences between the proposed parasitic-aware transistor sizing framework with prior work, is the use of graph neural network parasitic prediction during SPICE simulations and surrogate modeling. Parasitic prediction removes the need for in-the-loop layout generation when automated layout tools are limited and unstable. Furthermore, including parasitic information in the surrogate model greatly improves the accuracy and increases the convergence for Bayesian optimization.

B. Performance Modeling with Parasitic Graph Embedding

Conventional Bayesian optimization uses Gaussian process regression as the surrogate model. One major drawback of GPR-based Bayesian optimization is that model training time grows cubically with the number of observations. Recent work leveraged deep neural networks as learnable basis functions with Bayesian linear regression for improved scalability [19].

In this work, we propose an improved performance surrogate model using graph embeddings from the pre-trained parasitic graph neural network as additional parasitic information. Since both device sizing and critical parasitics can affect the circuit performance, we naturally want to include parasitic related information into the performance surrogate model. We leverage the pre-trained ParaGraph GNN model described in Sec. III to encode latent information of the circuit parasitics with graph embeddings. The parasitic GNN model is pre-trained with an abundant amount of data, including circuits with different topologies and sizing, while the performance model is targeted to a specific circuit.

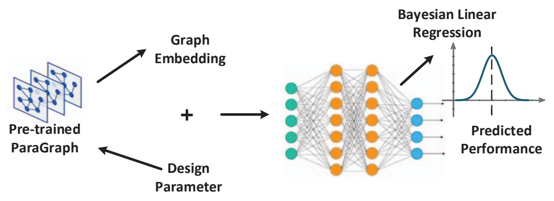


Fig. 4: Performance model with parasitic graph embedding.

Fig. 4 illustrates the proposed performance model with parasitic graph embedding. Given a specific design, the circuit graph is fixed. The node attributes input to the ParaGraph GNN model are related to the design parameters. The node embedding matrix $H = \{h_1 \cdots h_n\} \in \mathbb{R}^{n\times d}$ is obtained from the GNN by concatenating all n node embeddings of dimension d. The graph embedding g is then obtained by aggregating the node embedding matrix H, which includes information from both the design and parasitic prediction. Finally, the design parameter and graph embedding is fed into several layers of fully-connected neural networks for performance prediction. Similar to the work of [18], the last fully connected layer could be replaced with BLR to model uncertainty.

One crucial step is obtaining the graph embedding g from the node embedding matrix H. A simple method would be to use the mean of all node embeddings $g = \frac{1}{n}\sum_i^n h_i$. In practice, we found that introducing an additional self-attention layer [21] for weighted averaging improves the model results:

$$z_i = \sigma(w^T h_i) \tag{11}$$

$$w_i = softmax(z_i) = \frac{exp(z_i)}{\sum_1^n exp(z_i)} \tag{12}$$

$$g = \frac{1}{n}\sum_1^n w_i \cdot h_i, \tag{13}$$

where w is the self-attention weight of size d, and z_i, w_i are scalers. The intuition is that not all device and net nodes are critical and could have different importance.

C. Leveraging Dropout to Model Uncertainty

We propose leveraging dropout as an efficient and scalable method for predicting uncertainty. To estimate the predictive mean and predictive uncertainty we simply collect the results of stochastic forward passes through the model retained in training mode and compute their statistical mean and variance. Dropout is a commonly used method to reduce overfitting and improve generalization error for neural networks [22]. During training with dropout, the neurons are stochastically "dropped out" by sampling from a Bernoulli distribution. With modern GPU hardware, the forward passes can be done concurrently in batches, resulting in constant running time. Below, we provide both the Bayesian and frequentist perspective to the intuition of using dropout to predict uncertainty.

1) Bayesian: Dropout in deep neural networks could be seen as approximate Bayesian inference in deep Gaussian processes. Yarin et al. [23] provided a theoretical framework, where it is shown that a neural network with arbitrary depth and non-linearities, with dropout applied before every weight layer, is mathematically equivalent to an approximation to the probabilistic deep Gaussian process.

2) Frequentist: SMAC [24] leveraged random forest models to quantify uncertainty, and use the expected improvement as the acquisition function for sequential model-based optimization. Random forests are a collection of regression trees, thus the uncertainty is the variance between the prediction of different regression trees. Similarly, dropouts could be viewed as an economical approximation to training and using a very large ensemble of networks [25]. The computed statistical variance from the proposed GNN model in Sec. IV-B, could be viewed as sampled from both the predicted parasitic distribution, and ensembles of FOM prediction networks.

Since gradients for the predicted uncertainty with dropout are difficult to obtain, we use particle swarm optimization [26] (PSO) to maximize the acquisition function wEI in Sec. II-C with nevergrad toolbox [27]. We limit the number of budget queries (i.e. 10,000) and select the top-k (10) candidates for batched Bayesian optimization. In practice, we found PSO is a simple yet effective method to explore the design space more effectively since swarms of particles converge to different local maxima, and the selected candidates within a batch still maintain differences.

V. Experimental Results

The graph neural network models were implemented in Deep Graph Library [28], and the BLR and GPR models were based on Scikit-learn [29]. The models were trained on a dataset with a variety of analog and mixed-signal circuits in a 16nm finFET technology node on a NVIDIA Tesla V100 GPU with 16GB memory. In this work we only demonstrate results of extending parasitic prediction to coupling capacitance.

A. Improved Parasitic Prediction with Coupling Capacitance

TABLE I: Parasitic Simulation Comparison

	Stage1 Bandwidth (GHz)	Stage1 Gain @ Nyquist (dB)	Stage2 Bandwidth (GHz)	Stage2 Gain @ Nyquist (dB)
Post-Layout Extraction	32.2	9.40	11.2	17.4
Schematic (No Parasitics)	47.1	18.5	16.0	27.1
Designer Estimates	34.4	8.54	11.6	19.5
Paragraph Predicted C	68.1	12.63	15.6	18.9
Paragraph Predicted C+CC	31.4±0.2	10.4±0.4	9.8±0.7	17.0±0.3

The dataset for training parasitic prediction includes circuits from a 25Gb/s high-speed serial link design in a 16nm finFET technology node. It includes in 11 circuits with 8,613 devices selected for training and 3 circuits with 2,398 devices selected for testing. Optimization was performed on two of the designs selected for testing. We used a dropout rate of 0.3 for training. The lumped capacitance and coupling capacitance were able to achieve a testing R^2 score of 0.56 and 0.52.

To test the accuracy of the updated parasitic prediction model, important circuit metrics were simulated for a high-speed receiver amplifier test-circuit. Table I shows the circuit-performance summary using the parasitic estimates for lumped capacitance only (C) as well as the updated lumped parasitics with coupling capacitance estimates (C+CC). The mean and standard deviation in the results for C+CC were obtained using dropout (50 samples). Results are also included for the post-layout extracted netlist, schematic with no parasitic information and schematic with designer parasitic estimates. The updates to the parasitic prediction model (C+CC) reduced simulation error by 41.3% for these circuit metrics when compared with the lumped C results.

B. GNN Surrogate Performance Model

TABLE II: R^2 Score for Performance Model

Perf.	GNN-attn		GNN-BLR		GNN-mean		GPR	
	Train	Test	Train	Test	Train	Test	Train	Test
1	0.891	0.882	0.961	**0.931**	0.944	0.842	**0.974**	0.915
2	0.869	0.750	0.922	**0.780**	0.879	0.493	**0.930**	0.497
3	0.938	0.812	0.960	0.831	0.912	**0.852**	**0.994**	0.660
4	0.960	0.944	0.978	**0.965**	0.957	0.879	**0.999**	0.728
5	0.969	0.943	0.981	**0.954**	0.952	0.903	**0.999**	0.898
6	0.912	0.785	**0.942**	**0.801**	0.878	0.705	0.770	0.613
7	0.937	0.837	0.963	0.838	0.928	**0.854**	**0.999**	0.728
8	0.927	0.907	0.956	**0.916**	0.913	0.407	**0.993**	0.196
Avg.	0.925	0.858	**0.958**	**0.877**	0.920	0.742	0.957	0.654

We demonstrate the improvement of the GNN model with parasitic graph embedding proposed in Sec IV-B. Table II shows the R^2 score of predicting different performance metrics (objective and constraints) for the receiver amplifier on 500 designs (400 train 100 test). On average, the proposed GNN model outperforms GPR by more than 20%. The compared models are as follows:

- **GNN-attn**: GNN model proposed in Sec IV-B with self-attention graph embedding readout.
- **GNN-BLR**: Replace last layer of GNN-attn with BLR.
- **GNN-mean**: GNN model proposed in Sec IV-B with taking the mean node embedding as graph embedding.
- **GPR**: GPR model with only the design parameters.

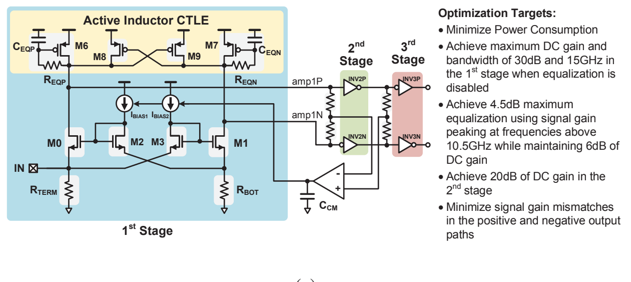
C. Parasitic-Aware Circuit Sizing

We demonstrate the proposed sizing framework on two designs. The two designs are selected from the testing set for training the parasitic prediction model. Fig. 5 shows the circuit schematics and optimization targets. The design parameters are carefully selected to cover specific design targets, while certain device sizes are constrained based on the relative sizing of other devices. For all experiments, we first obtain 30 random designs and then use Bayesian optimization.

1) Receiver Amplifier: The receiver front-end is comprised of a 3-stage amplifier. The first stage is highly configurable through digital control of device sizes to provide tunability of the equalization (EQ) configurations. The design parameters consists of 10 device parameters (8 discrete).

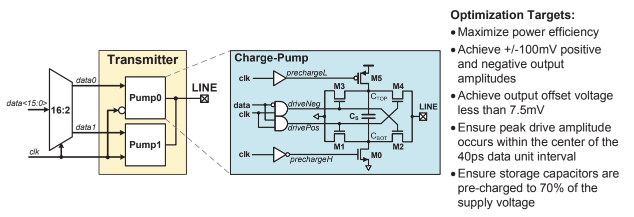
2) Transmitter Line-Driver: The transmitter consists of an output-multiplexed pair of charge-pumps as the line driver. Each of the charge-pumps operate out-of-phase and alternate between a pre-charge and line-drive phase. The design parameters consists of 7 device parameters (6 discrete).

Fig. 6 shows the optimization convergence. The cost is calculated as the sum of normalized performance with constraints penalized by 10 times more than the objective, averaged by the top 20 candidates. The compared surrogate models are:



Optimization Targets:
- Minimize Power Consumption
- Achieve maximum DC gain and bandwidth of 30dB and 15GHz in the 1st stage when equalization is disabled
- Achieve 4.5dB maximum equalization using signal gain peaking at frequencies above 10.5GHz while maintaining 6dB of DC gain
- Achieve 20dB of DC gain in the 2nd stage
- Minimize signal gain mismatches in the positive and negative output paths

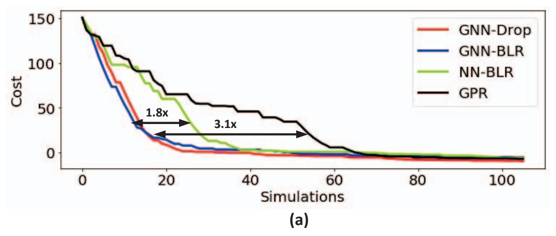
(a)



Optimization Targets:
- Maximize power efficiency
- Achieve +/-100mV positive and negative output amplitudes
- Achieve output offset voltage less than 7.5mV
- Ensure peak drive amplitude occurs within the center of the 40ps data unit interval
- Ensure storage capacitors are pre-charged to 70% of the supply voltage

(b)

Fig. 5: Circuit Schematics. (a) Receiver amplifier (b) Transmitter line-driver



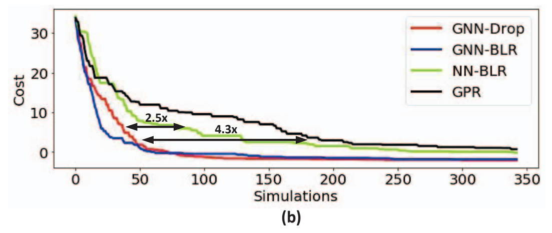


Fig. 6: Optimization convergence. (a) Receiver amplifier (b) Transmitter line-driver

- **GNN-Drop**: GNN model using dropout for uncertainty
- **GNN-BLR**: GNN model with last layer as BLR
- **NN-BLR**: Neural network with last layer as BLR on only design parameters as in [18]
- **GPR**: GPR model on only design parameters

It could be observed that for both designs, the improved GNN model converges faster than the baselines models that do not use parasitic graph embedding. On average, the improved surrogate model with graph embedding input helps convergence by 2.1 times speedup compared to BLR without GNN, and 3.7 times speedup to GPR. Furthermore, **GNN-Drop** has similar convergence as **GNN-BLR**, demonstrating that dropout is an effective measure of uncertainty.

Table III and IV demonstrate the final obtained optimization results and the design objective and constraints. It also compares the designs obtained by excluding parasitic prediction

TABLE III: Receiver Amplifier Optimization Results

Amplifier Configuration	Performance Metric	Units	Objective	Manual Design	Opt. w pred.	Opt. w/o pred.	
						Sim. w/o pred.	Sim. w pred.
-	Supply Current	mA	min	3.55	3.57	3.65	3.65
Minimum Peaking	Equalization Strength	dB	$<$0.5	0.48	0.49	0.3	4.8
	DC Gain	dB	$>$30	33.7	36.7	35.8	35.6
	Bandwidth	GHz	$>$15	18	17	20.6	6.0
Maximum Peaking	Equalization Strength	dB	$>$4.5	5.0	6.1	5.4	7.5
	DC Gain	dB	$>$6	7.7	9.5	8.86	8.83
	Peak Freq.	GHz	$>$10.5	12.7	12.6	17.8	2.8
-	Stage2 DC Gain	dB	$>$20	20.9	20.3	21.21	21.08
-	+/- Gain Mismatch	dB	$<$2	0.8	0.6	0.87	0.95

Manual Design Sized manually with completed layouts, simulated with parasitic extraction.
Opt. w pred. Optimize with parasitic prediction. *Opt. w/o pred.* Optimize with only schematic.
Sim. w/o pred. Simulated performance without parasitic prediction. *Sim. w pred.* Simulated performance with parasitic prediction.

TABLE IV: Transmitter Line-driver Optimization Results

Performance Metric	Units	Objective	Manual Design	Opt. w pred.	Opt. w/o pred.	
					Sim. w/o pred.	Sim. w pred.
Average Efficiency	-	max	0.35	0.32	0.45	0.28
Positive Amplitude	V	$>$0.1	0.12	0.11	0.12	0.12
Negative Amplitude	V	$>$0.1	0.10	0.10	0.11	0.09
Offset Voltage	mV	$<$7.5	5.28	7.50	4.75	12.0
Positive Drive Peak Time	ps	$\in$ (17,23)	18	21	18	22
Negative Drive Peak Time	ps	$\in$ (17,23)	21	21	20	25
Pre-Charge Voltage	V	$>$0.5	0.52	0.52	0.64	0.56

in the optimization loop. Designs optimized without parasitics result in numerous constraints failing (denoted with red text) when re-simulated with predicted parasitics.

D. Scalability and Training Runtime

Figure 7 shows the normalized training runtime for different models. The runtime is normalized with training the smallest data size (30) for comparing the scalability. In theory, the BLR and GNN model should have $O(N)$ runtime complexity, while GPR is $O(N^3)$ [18]. GPR scales poorly and its absolute runtime quickly out-scales the BLR and GNN.

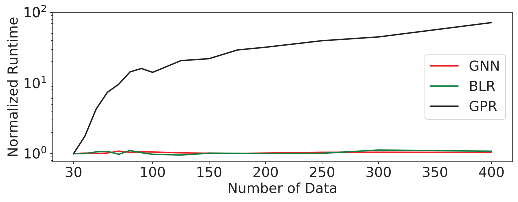


Fig. 7: Normalized training time for different models.

VI. Conclusion

In this work, we present a parasitic-aware sizing framework leveraging recent developments in parasitic prediction with graph neural networks. We present the method on extending prior work to include parasitic resistance and coupling capacitance prediction. We further propose an improved surrogate model with parasitic graph embedding and leverage dropout to predict uncertainty for Bayesian optimization. Future work includes continued research of using graph learning for analog design optimization.

References

[1] G. Gielen, H. Walscharts, and W. Sansen, in *ESSCIRC*, 1989.

[2] M. d. Hershenson, S.P. Boyd, and T.H. Lee, "Optimal design of a cmos op-amp via geometric programming," *IEEE TCAD*, 2001.

[3] Bo Liu, F.V. Fernandez, and G. Gielen, "Fuzzy selection based differential evolution algorithm for analog cell sizing capturing imprecise human intentions," in *IEEE Congress on Evolutionary Computation*, 2009.

[4] W. Lyu, P. Xue, F. Yang *et al.*, "An efficient bayesian optimization approach for automated optimization of analog circuits," *IEEE Transactions on Circuits and Systems I*, 2018.

[5] M. Liu, K. Zhu, X. Tang *et al.*, "Closing the design loop: Bayesian optimization assisted hierarchical analog layout synthesis," in *DAC*, 2020, pp. 1–6.

[6] H. Wang, K. Wang, J. Yang *et al.*, "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *DAC*, 2020.

[7] M. Ranjan, W. Verhaegen, A. Agarwal *et al.*, "Fast, layout-inclusive analog circuit synthesis using pre-compiled parasitic-aware symbolic performance models," in *DATE*, 2004.

[8] H. Habal and H. Graeb, "Constraint-based layout-driven sizing of analog circuits," *IEEE TCAD*, 2011.

[9] N. Lourenço, R. Martins, and N. Horta, "Layout-aware sizing of analog ics using floorplan routing estimates for parasitic extraction," in *DATE*, 2015.

[10] K. Hakhamaneshi, N. Werblun, P. Abbeel *et al.*, "BagNet: Berkeley analog generator with layout optimizer boosted with deep neural networks," in *ICCAD*, 2019.

[11] K. Settaluri, A. Haj-Ali, Q. Huang *et al.*, "AutoCkt: Deep reinforcement learning of analog circuit designs," in *DATE*, 2020.

[12] J. Crossley, A. Puggelli, H. Le *et al.*, "BAG: A designer-oriented integrated framework for the development of ams circuit generators," in *ICCAD*, 2013.

[13] H. Ren, G.F. Kokai, W.J. Turner *et al.*, "ParaGraph: Layout parasitics and device parameter prediction using graph neural networks," in *DAC*, 2020.

[14] B. Shook, P. Bhansali, C. Kashyap *et al.*, "MLParest: Machine leaning based parasitic estimation for custom circuit design," in *DAC*, 2020.

[15] K. Kunal, J. Poojary, T. Dhar *et al.*, "A general approach for identifying hierarchical symmetry constraints for analog circuit layout," in *ICCAD*, 2020, pp. 1–8.

[16] X. Gao, C. Deng, M. Liu *et al.*, "Layout symmetry annotation for analog circuits with graph neural networks," in *ASPDAC*, 2021.

[17] Y. Li, Y. Lin, M. Madhusudan *et al.*, "A customized graph neural network model for guiding analog ic placement," in *ICCAD*, 2020.

[18] S. Zhang, W. Lyu, F. Yang *et al.*, "Bayesian optimization approach for analog circuit synthesis using neural network," in *DATE*, 2019.

[19] J. Snoek, O. Rippel, K. Swersky *et al.*, "Scalable bayesian optimization using deep neural networks," *Proceedings of Machine Learning Research*, 2015.

[20] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *J. Am. Soc. Inf. Sci. Technol.*, 2007.

[21] A. Vaswani, N. Shazeer, N. Parmar *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017.

[22] N. Srivastava, G. Hinton, A. Krizhevsky *et al.*, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, 2014.

[23] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of Machine Learning Research*, 2016.

[24] F. Hutter, H.H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International Conference on Learning and Intelligent Optimization*, 2011.

[25] P. Baldi and P.J. Sadowski, "Understanding dropout," in *Advances in Neural Information Processing Systems*, 2013.

[26] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *International Conference on Neural Networks*, 1995.

[27] J. Rapin and O. Teytaud, "Nevergrad - A gradient-free optimization platform," https://GitHub.com/FacebookResearch/Nevergrad, 2018.

[28] M. Wang, D. Zheng, Z. Ye *et al.*, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, 2011.