Fully Automated Machine Learning Model Development for Analog Placement Quality Prediction

Chen-Chia Chang[1], Jingyu Pan[1], Zhiyao Xie[2], Yaguang Li[3], Yishuang Lin[3], Jiang Hu[3], and Yiran Chen[1]
Duke University[1] Hong Kong University of Science and Technology[2] Texas A&M University[3]
{chenchia.chang, jingyu.pan, yiran.chen}@duke.edu, eezhiyao@ust.hk

ABSTRACT

Analog integrated circuit (IC) placement is a heavily manual and time-consuming task that has a significant impact on chip quality. Several recent studies apply machine learning (ML) techniques to directly predict the impact of placement on circuit performance or even guide the placement process. However, the significant diversity in analog design topologies can lead to different impacts on performance metrics (e.g., common-mode rejection ratio (CMRR) or offset voltage). Thus, it is unlikely that the same ML model structure will achieve the best performance for all designs and metrics. In addition, customizing ML models for different designs require more tremendous engineering efforts and longer development cycles. In this work, we leverage Neural Architecture Search (NAS) to automatically develop customized neural architectures for different analog circuit designs and metrics. Our proposed NAS methodology supports an unconstrained DAG-based search space containing a wide range of ML operations and topological connections. Our search strategy can efficiently explore this flexible search space and provide every design with the best-customized model to boost the model performance. We make unprejudiced comparisons with the claimed performance of the previous representative work on exactly the same dataset. After fully automated development within only 0.5 days, generated models give 3.61% superior accuracy than the prior art.

1 INTRODUCTION

The performance of analog IC largely depends on the circuit layout quality. Despite decades of research towards automated layout generation [7, 9, 10, 17], analog layout in practice is still a manual and time-consuming process, leading to a high engineering cost and long time-to-market. Many traditional analog placement methods enforce certain heuristic constraints without explicit evaluations on post-layout circuit performance. These heuristic constraints cannot precisely capture complex correlations between layout and circuit performance for different scenarios. Thus, providing an efficient and effective way to estimate layout performance is important while optimizing analog placement solutions.

In recent years, various machine learning (ML) approaches [8, 11] have been developed to explicitly predict the final post-layout performance. These methods are applied in various EDA applications such as pre-routing design space exploration [11] and guiding the placement process [8]. Liu *et al.* [11] proposed a well-designed convolution neural network (CNN) model to capture the complex correlation between the layout and final performance. Li *et al.* [8] developed a customized graph neural network model to estimate the impact of placement on layout performance. However, existing methods typically relied on developers to manually design one ML model architecture to predict for all designs and for all performance metrics. Considering the vast difference among analog designs, one fixed ML architecture, regardless of how sophisticated it is, cannot fit the requirements on all analog designs for different metrics, thus leading to sub-optimal performance. More importantly, a fixed ML model would suffer from reliability problems due to possible accuracy degradation when applied towards new designs.

In comparison, allowing customized ML models for various designs and performance metrics is a straightforward yet promising way to achieve superior overall performance and more reliable predictions. However, such model customization is constrained by the manpower budget of design companies. Considering a typical ML model development process in design flow easily takes weeks [2], further customizing ML architectures for different analog designs will take prohibitively tremendous engineering efforts and a long turnaround time. A recent work [2] proposed to automatically develop ML estimators for digital designs with NAS techniques. However, this work only focused on the digital routability estimation problem, a small EDA field.

To reduce the engineering effort on ML model development for chip design, we extend the automated ML model development framework [2] to analog design, targeting analog placement quality prediction. This is achieved by leveraging NAS, which efficiently searches the most appropriate customized model for each task from a comprehensive search space. Our approach achieves superior performance than representative prior work [11] through *unprejudiced* comparisons. As a result, our work demonstrates the extraordinary generalization ability of the framework. Such generalization ability applies to various prediction tasks for digital and analog designs, including the previous routability estimator and this analog placement quality model development. Thus, we believe this general automation framework has great potential for an extensive range of design problems. Our main contributions in this work are summarized as follows:

- We construct a generic framework that supports automated placement quality estimators development without any human interference. This work is the first exploration of automated ML model development for analog design to the best of our knowledge.
- Our method supports a large search space allowing various operations and highly flexible connections, accommodating solutions beyond the scope of human-designed models.
- The automatically developed models outperform state-of-the-art layout performance estimators [11] through *unprejudiced* comparisons Evaluations are performed on exactly the same open-sourced datasets [11], directly adopting their claimed performance as baselines.
- The whole model search process of our NAS-based framework takes only 0.5 days, while the human developers easily spend weeks to months in model development. In addition, we conduct a detailed analysis on automatically developed models, which significantly differs from human-developed estimators. We believe this work will benefit the development of layout estimators in the future.

The rest of the paper is organized as follows. Section 2 introduces the background of our target task, the analog placement quality prediction. Section 3 formulates our model customization problem for the analog placement quality prediction. Section 4 details our automatic ML model development method. Section 5 demonstrates the experimental results, and Section 6 further discuss our proposed framework. Finally, Section 7 concludes this paper.

ASPDAC '23, January 16–19, 2023, Tokyo, Japan

ACM ISBN 978-1-4503-9783-4/23/01…$15.00
https://doi.org/10.1145/3566097.3567881

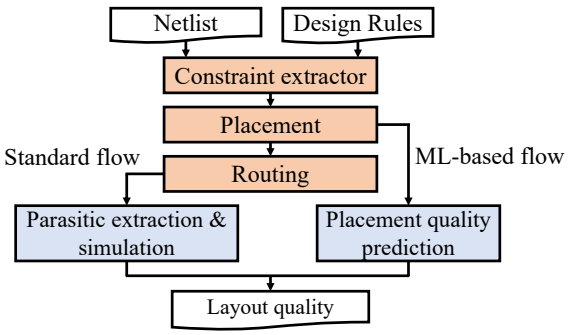

Figure 1: Analog IC design automation and quality evaluation flow. A netlist and design rules comes into the constraint extractor, the placement, and the routing stage to produce the physical layout. In the standard evaluation flow, we use parasitic extraction and simulation to obtain layout quality. With ML model, we can directly use the placement result to predict layout quality.

2 BACKGROUND OF ANALOG AUTOMATION

Analog IC design automation flow aims to automatically generate physical layouts from circuit netlists. In recent years, some frameworks are developed [6, 16] to achieve analog IC design automation. The layout design flows of both works [6, 16] contain the placement and routing (P&R) stages. An example of the analog IC design automation flow proposed in [16] is shown in Figure 1. A netlist will first come through the constraint extractor and the P&R stage to generate the layout. In the P&R stage, however, adopting the same objective used for the digital circuit, e.g., wirelength and area minimization, cannot guarantee the layout solution quality because the analog layout quality is affected by the other geometric factors such as symmetric and common-centroid relations [12]. Many automatic analog placement [12, 14] and routing [15, 18] methods have been proposed to meet geometric relations but still are lack of a proper model to measure the layout quality. Thus, a precise layout quality estimation in the P&R stage is desirable.

The work [11] proposes an ML-based approach to perform placement quality prediction, which takes the placement result as an input and predict the layout quality generated by the parasitic extraction and simulation, as shown in Figure 1. In [11], they view their placement results as two-dimensional images and apply CNN model to predict placement quality. Their approach uses one general-purpose structure to predict different layout quality metrics on different analog designs. Due to the topology heterogeneity between different analog designs, one model structure may not be well-suited for all designs. Thus, the model customization is needed for each design and quality metric to achieve promising prediction performance. As a result, in this work, we introduce NAS to automate the ML model customization. In this way, we can conduct the customization without additional engineering efforts and can accelerate the customization process. We use a highly flexible model search space that contains a variety of diverse model structures, enabling us to find the best model for each analog design. In addition, our search process only takes about 0.5 day to find a promising model for a analog design. With a short search period, we can make a great prediction performance improvement by our automatic model customization method.

3 PROBLEM FORMULATION

This work applies NAS techniques to automate designs of ML models for analog layout performance prediction. After placement, a layout is sliced into $w \times h$ tiles and transformed to input feature $X_i \in \mathbb{R}^{w \times h \times c}$, which is comprised of c different two-dimensional feature maps. The ground-truth layout performance y_i are collected after the subsequent routing, parasitic extraction, and SPICE simulation. The type of y_i includes offset voltage and CMRR. Finally, following [11], we separate our placement solutions into 75% good and 25% bad layouts based on y_i and formulate y_i into binary classification. Based on the extracted features and label, the analog layout performance prediction task can be formulated below:

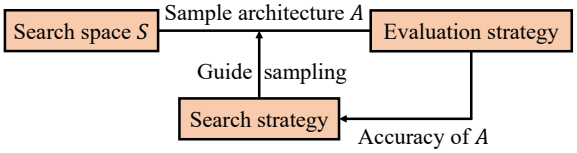

Figure 2: Overview of Neural Architecture Search. We first sample an architecture from the search space and estimate its accuracy by our evaluation strategy. With the search strategy, we can control the sampling according to the accuracy.

Problem. *Given a set of placement solutions with the input feature X_i, the defined search space S_C, and y_i, it aims to explore the architecture $A_C \in S_C$ of the neural network model f_{A_C} to predict y_i such that the performance of f_{A_C} is maximized, where*

$$f_{A_C} : X_i \in \mathbb{R}^{w \times h \times c} \rightarrow y_i \in \{0, 1\}.$$

Thus, our NAS approach aims to generate customized models to maximize the model performance on different designs and metrics.

4 TOWARDS AUTOMATED MACHINE LEARNING MODEL DEVELOPMENT

To achieve automated ML model development, we leverage NAS techniques to explore high-performance customized models for analog layout performance prediction for all analog circuit designs without human interference. Typically, NAS contains three key ingredients: *search space*, *evaluation strategy*, and *search strategy*. Search space defines a space of candidate model architectures that can be explored in NAS. Evaluation strategy determines how to estimate the design metrics (e.g., accuracy) of a candidate architecture and provides feedback to the search process. Search strategy is the method to explore the search space and guide the search process toward a correct selection of the promising ML model. The overall procedure of NAS is sketched in Fig. 2.

We introduce a graph-based NAS method to automate the design of neural networks, following the spirit of SwiftNet [4]. We start with introducing our graph-based search space, where a neural architecture can be easily partitioned into two parts: fixed and searchable parts composed of Directed Acyclic Graphs (DAGs). While the fixed parts are responsible for performing standard transformations, e.g., downsampling, to process the features, the searchable blocks are capable of formulating a high-quality learned representation to extract critical information from the input features, leading to improved performance on the placement quality prediction. Then, we present our graph propagation strategy that efficiently harness our graph-based search space to discover promising neural architectures within limited search budget. Finally, we disclose the evaluation strategy and defines the criteria on evaluating different candidate models within the graph-based search space.

4.1 Graph-based Search Space

We leverage CNN as our main architecture of interest in placement quality prediction. We can view CNN as a graph constructed by a set of operations and the connections of operations. Specifically, vertices represent operations, and edges indicate the directed connections of operations to pass tensors from one vertex to another. The illustration of our graph-based search space is shown in Figure 3. For the searchable part, we employ 6 DAGs (i.e. $\{G_1, G_2, \ldots, G_6\}$), named *guide-DAGs*, to represent the overall search space that admits a wide set of possible CNN architectures. Each guide-DAG $G_i(V_i, E_i)$ represents a combination of

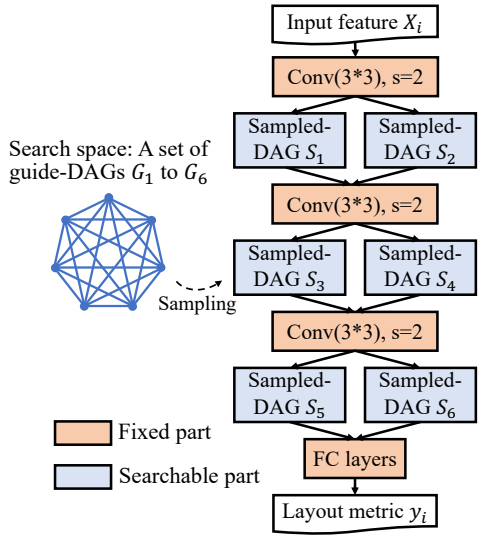

Figure 3: The search space and the sampled architecture of our NAS method. Our model architecture is separated into the searchable and fixed part. The searchable part composes of six sampled-DAGs, whose structures are sampled from the guide-DAGs. The fixed part has three the convolution layers with kernel size 3 and stride 2 to perform downsampling and two fully-connected layers to produce the layout metric.

the candidate operations and the propagation of data tensors. Specifically, a guide-DAG is composed by a set of completely ordered vertices V_i with maximum edges to provide all possible connections and with each vertex $v \in V_i$ representing a candidate operation OP_v. During our graph-based NAS, we sample sub-graphs from the guide-DAG to obtain different neural architectures, and name the sampled graphs as sampled-DAGs (i.e. $\{S_1, S_2, \ldots, S_6\}$). For the fix part, we use 3 convolution layers with kernel size 3 and stride 2 to downsample the input features. Then, in the head stage, we use a mean pooling layer and two fully-connected layers with size 32 and 1 to produce the output layout metrics. Combining both searchable and fix parts, we can construct our sampled model architecture. We elaborate the vertex and the edge operation as follows.

Vertex Operation. Each vertex obtains the source of input concatenates all input tensors from the incoming edges and produces the output tensor by a given c operation OP_v. To ensure a good performance on placement prediction, we review prior literature in computer vision and carefully select the following candidates as vertex operations:

- 3×3 convolution with 32 filters
- 3×3 convolution with 64 filters
- 3×3 atrous convolution with 32 filters, dilation rate 2
- mixed convolution with 4 groups, kernel size $[7, 9, 11, 13]$

First, we include regular convolution layers with different numbers of filters as our operation option. Then, we observe that atrous convolution [3] can effectively enlarge receptive fields of filters thus help model to capture large regions of relations between different transistors. In addition, we choose a mixed depth-wise convolution operation (MixConv) [13] to separate channels into groups and apply different kernel sizes to each group. As such, we can identify transistor placement patterns in different sizes when applied in this layout quality prediction. As a result, the high variety of operations improves our model diversity and cover much more potential high-quality models in the search space.

Edge Operation. Each edge $e(u, v) \in E_i$ represents the propagation of the output tensor of vertex u to the input of v. Here, E_i is constructed to make G_i with maximum edges to provide all possible connections in

Algorithm 1 *Graph Propagation*

Require: $G_i(V_i, E_i)$
Ensure: $S_i(V_{S_i}, E_{S_i})$
1: $V_{S_i} = \{v_0\}, v_0 \in V_i$
2: $E_{S_i} = \emptyset$
3: **for** each $v_j \in V_i$ **do**
4: **if** $v_j \in V_{S_i}$ **then**
5: **for** each $e(v_j, u_k) \in E_i$ **do**
6: $p = \frac{\exp(w_{e(v_j,v_k)})}{\sum_{l=j}^{7} \exp(w_{e(v_j,v_l)})}$
7: random $r(0, 1)$
8: **if** $p > r$ **then** ▷ sampled by probability p
9: $V_{S_i} = V_{S_i} \cup \{v_k\}$
10: $E_{S_i} = E_{S_i} \cup \{e(v_j, v_k)\}$
11: **for** each $v_j \in V_{S_i}$ **do**
12: random $r(0, 1)$
13: $p = 0$
14: **for** each $w_{OP_k v_j}, k = 1$ to 4 **do**
15: $p' = \frac{\exp(w_{OP_k v_j})}{\sum_{l=i}^{4} \exp(w_{OP_1 v_j})}$
16: $p = p + p'$
17: **if** $p > r$ **then** ▷ sampled by probability p'
18: $OP_{v_j} = OP_k$
19: break
20: **return** $S_i(V_{S_i}, E_{S_i})$

the sampled-DAG. Our search space develops many parallel propagation of tensors within a sampled-DAG becuase concatenation and parallel propagation of tensors can help the model to facilitate feature combinations between vertices to discover different layout quality information. In addition, the parallel sampled-DAG structures between every two downsampling layers can also explore different feature representations of layout quality.

Thanks to the great flexibility of connections and operations, our graph-based search space can produce the high-quality model architecture for layout performance prediction. As a result, our NAS methodology can explore diverse customized models to fit different analog designs and layout performance metrics based on this flexible search space.

4.2 Graph Propagation Strategy

Due to the large size of our graph search space, it is neither efficient nor practical to examine every sub-graph and every combination of operations. For example, it is impossible for random search to obtain a well-performance model in an efficient manner. Thus, designing a proper search strategy with this large and high flexibility search space is crucial.

We propose graph propagation, an efficient search strategy via sampling and continuous weight updating. Specifically, our graph propagation composes of two stages: 1) Edge sampling and 2) operation sampling. Edge sampling selects edges to form the sampled architecture, and operation sampling decides the internal operation within each selected vertex. Note that different from the work [2], we add the operation sampling method to enhance our search strategy. During each search iteration, we assign a weight to each edge and each vertex operation that control its sampling probability and then increase weights of the promising edges and vertex operations that lead to better architectures. During our search process, we will gradually update weights based on the sampled model performance to find a promising model.

We demonstrate the graph propagation algorithm in Algorithm 1. For edge sampling stage (Line 1-10), we first initialize the vertex set of sampled-DAG V_{S_i} with v_0 (lines 1), the vertex that represents the downsampling convolution layer. Because v_0 is in the fixed part, we view it as the sampling start vertex. Then, we iterate through each vertex $v_j \in V_i$. If v_j is selected in V_{S_i}, for each outgoing edge $e(v_j, v_k)$ of v_j, we

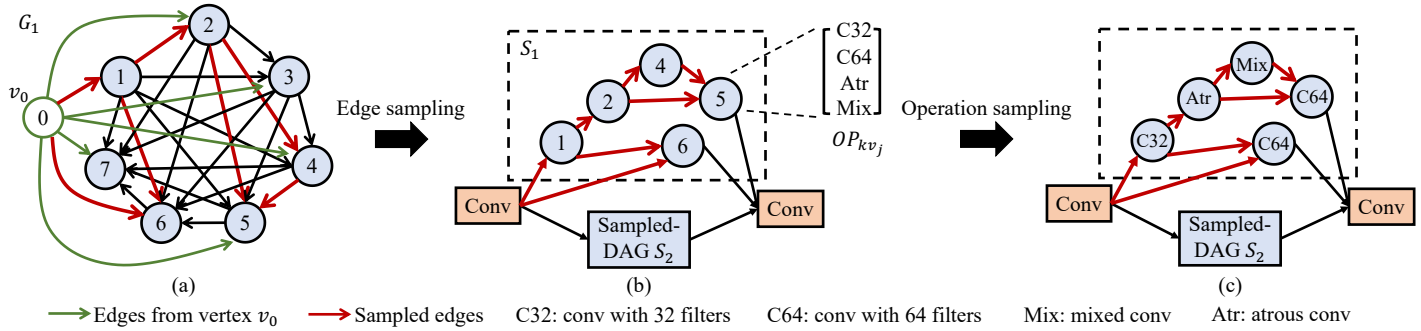

Figure 4: An example of graph propagation. For edge sampling, we select edges from the guide-DAG G_1 (a) to form the sampled-DAG S_1 (b). Then, we choose the operations of the selected vertices in operation sampling to form the sampled architecture (c).

normalize its weight to be its edge selection probability p (Lines 3-6). Note that a larger weight means a higher probability to be sampled. We apply softmax to perform normalization, which can enhance and reflect the difference between weights on the probability. If $e(v_j, v_k)$ is selected, we add $e(v_j, v_k)$ and v_k into our sampled DAG (Lines 8-10). In later iterations, the outgoing edges of v_k will be extracted and performed sampling. After sampling all the vertices and edges of S_i, we enter the operation sampling stage (Line 11-19) to select the operation of each vertex $v_j \in V_{S_i}$ (Line 11). We also utilize softmax to normalize the operation weights of v_j. The normalized weights directly represent the selection probabilities, and we choose one operation for each v_j (Lines 12-19). Finally, S_i is returned (Line 20) after selecting all the operations of vertices. An example of graph propagation is illustrated in Figure 4. Note that v_0 represents the fixed convolution layer before S_1, and for the nodes without sampled outgoing edge (5 and 6), we will directly connect it to the subsequent fixed convolution layer.

After sampling each S_i, we construct our model through the architecture shown in Figure 3 and measure the accuracy based on our evaluation strategy, which will be discussed Section 4.3. According to the accuracy η, for each G_i, we update weights w_e of edges e that are selected into S_i in this iteration based on the below equation:

$$w_e = w_e * \exp(\alpha(\eta - \beta)),$$

where α is the updating rate, and β is the baseline accuracy. We update weights according to the difference between η and β and apply an exponential function to accelerate the weight update. In addition, we update the operation weight w_{OP_v} in the same way of w_e. When we get a higher accuracy than the baseline, the edge and vertex operation weights will increase. Thus, in the next iteration, the sampled probabilities of these edges and operations will grow. Also, the baseline metrics β is set to the average accuracy of all history sampled models. In this way, we can prompt the search process to seek higher-performance models through iterations. Finally, this search process, including sampling and weight updating, continues until our model performance converges.

As graph propagation covers every sub-graph in our search space, it makes the number of vertices and edges varied in each sampled-DAG. In addition, graph propagation can explore every vertex operation combination. As a result, graph propagation enables a highly flexible choice of our sampled model and provides a higher chance of obtaining state-of-the-art architectures for difference designs. Despite the great flexibility, the search cost for graph propagation is only 0.5 days, indicating its remarkable efficiency. Thus, graph propagation can efficiently craft suitable customized models for analog designs even with huge diversity.

4.3 Evaluation Strategy

Following the previous NAS work [1], we directly train our sampled model on the training split of our target dataset and evaluate the accuracy on the validation split. The evaluation result is directly employed as our search objective.

Table 1: Data Statistics of [11].

Design (#transistor)	Compensation	Layouts	Metrics
OTA1 (31)	Nested Miller	16376	Offset
OTA2 (31)	Nested Miller	16381	Offset
OTA3 (22)	Miller	16384	Offset
OTA4 (23)	None	16363	CMRR

5 EXPERIMENTAL RESULTS

In this section, we first describe our experiment setups. We then present our *unprejudiced* comparison results with the previous work on exactly the same training and testing benchmark.

5.1 Experiment Setup

The baseline method is a well-designed CNN-based estimator 3D-CNN [11]. To have a fair comparison, we directly take the claimed accuracy presented in their work to compare with our NAS method. we validate our method using the same open-sourced dataset and the same setting as the work [11]. The dataset includes all OTA (Operational Transconductance Amplifier) designs but with different topologies and compensations. The summarized data statistics about this dataset is shown in Table 1. Note that OTA1 and OTA2 are in the same schematic but with different sizing. Their layout performance is evaluated by offset voltage for OTA1 to 3 and CMRR for OTA4. For each circuit design, the dataset is separated into 80% for training and 20% for testing. For labeling, we also follow the practice of the previous work, which formulates the problem into binary classification by given thresholds. For the feature extraction before training, we also follow the feature construction method in [11], in order to only focusing on the influence of model structure.

We adopt the NAS method described in Section 4 to explore the search space defined for layout quality prediction. The overall search process runs for 0.5 days on a NVIDIA TITAN RTX GPU with Intel® Xeon® E5-2687W CPUs. In comparison, an ML expert typically spends weeks to months to design a promising model [2]. Obviously, our NAS process largely shortens the development cycle of ML models for analog automation.

We employ the following hyperparameters to conduct model training in our experiments: we train our model for 45 epochs with Adam optimizer [5], a batch size of 48, and a fixed learning rate of 0.0005. To combat overfitting and improve generalization, we use an L2 weight decay of 10^{-3} and ReLU activation.

5.2 Layout Performance Comparison

In Table 2, we compare the performance of NAS-crafted models with the claimed performance of CNN [11]. Note that our NAS flow automatically generates customized models for different OTA designs separately. Our NAS-crafted models clearly achieve higher accuracy on all designs with average 1.85% higher accuracy. More importantly, our NAS-crafted

Table 2: Placement quality prediction result with dataset [11].

Design (#transistor)	Accuracy ↑		
	CNN [11]	NAS-crafted	Improvement
OTA1 (31)	90.29	**91.39**	1.22%
OTA2 (31)	92.61	**92.74**	0.14%
OTA3 (22)	91.33	**93.57**	2.45%
OTA4 (23)	92.05	**95.37**	3.61%
Avg	91.57	**93.27**	1.85%

models reach maximum 3.6% improvement in accuracy. These results indicate that automatically customizing models for different designs with our NAS method can effectively boost the model performance without any designer's manual interference. Most importantly, the search process only takes about 0.5 days to achieve significant performance improvement.

In addition, we validate our NAS-crafted models using the same transfer learning scheme of [11]. We utilize transfer learning to trained s model on one design and then finetune this model on another design. Following [11], we first pretrain the model with OTA1 design using $\alpha = 0.8$ ratio of all data, then finetune on other designs with different α. With the transfer learning scheme, we can reduce the data usage of other designs and get the comparable results of training on the full dataset. Reducing the data usage is important because it can shorten the data generation time and can also accelerate the model development period. Note that $\alpha = 0.8$ means using the entire set of training data because the data is assigned 80% for training. In addition, $\alpha = 0$ means we only perform inference on the target design without any finetuning.

In Table 3, our NAS-crafted models can outperform CNN [11] in all the cases with average 8.59% higher accuracy. It shows our customized models also perform well when transferring to other designs. In addition, as α decreases from 0.8 to 0.01, the degradation of our model performance is not as large as CNN [11], which shows that our model has great data-efficiency. Also, the average improvement ratio in transfer learning scheme is larger than the one in Table 2, which indicates that our NAS flow not only can find high-performance model for each specific design but also can give a better starting point in transfer learning scheme.

6 DISCUSSION

In this section, we first give an analysis of our automatic model customization scheme. Then, we compare our NAS-crafted models with different designs and try to provide insights about the analog layout performance prediction based on our NAS-crafted models.

Effectiveness of our automatic model customization. We propose a NAS framework to automatically customize ML models for different designs and layout quality metrics. To examine the effectiveness of this customization scheme, we take the NAS-crafted model searched for each design and then let it train on OTA1 and OTA3 data from scratch. We select OTA1 and OTA3 as our case study because two designs have highly different transistor numbers. According to Table 4, based on OTA1 data, the model searched for OTA1 can outperform other models. This observation also can be observed in OTA3. Thus, this comparison demonstrates that our automatic model customization can find the model that can achieve superior model performance on its target design.

Comparison of our NAS-crafted models for different designs. First, we compare and analyze the NAS-crafted models for OTA1 and OTA3, whose both layout metrics is the offset voltage but with different transistor numbers. Figure 5 (a) and (b) sketch our NAS-crafted models for OTA1 and OTA3, respectively.

In sampled-DAG 1 and 2, we observe OTA1 model utilizes wider and more convolution layers than the OTA3 model, which can help the design with more transistors explore richer feature representations. In

Table 3: Transfer learning result with dataset [11].

Design	α	Accuracy ↑		
		CNN [11]	NAS-crafted	Improvement
OTA1	0.8	90.29	**91.39**	1.22%
OTA2	0.8	90.96	**92.22**	1.39%
	0.1	90.10	**92.43**	2.59%
	0.01	88.28	**92.16**	4.40%
	0.0	70.10	**83.52**	19.14%
OTA3	0.8	90.23	**91.05**	0.91%
	0.1	87.29	**91.15**	4.42%
	0.01	81.21	**87.84**	8.16%
	0.0	74.73	**75.01**	0.37%
OTA4	0.8	89.91	**95.31**	6.01%
	0.1	88.70	**94.79**	6.87%
	0.01	90.10	**92.86**	3.06%
	0.0	49.72	**76.16**	53.18%
Avg		83.20	**88.91**	8.59%

sampled-DAG 3 and 4, both models use the most operations compared to the other sampled-DAGs in their own model. Thus, this stage is the most important one to extract some effective features. In this stage, the input tensor size is reduced by 4x. We speculate this is the appropriate resolution for these models to identify some impacts of placement relations between different transistors. In addition, this stage uses more mixed convolutions and atrous convolutions. Since both mixed convolution and atrous convolution have large receptive fields, more placement features in a wide layout region can be captured at this stage. In sampled-DAG 5 and 6, the OTA1 model still needs some large receptive field convolution operations to enrich the representations before passing it to the head stage. On the other hand, the OTA3 model just passes the output extracted by the previous stage to the final head stage. These observations show that models for large designs require more operations in the last stage to produce the final prediction on the offset voltage. In addition, we observe the sampled-DAG 5 and 6 of OTA1 and the sampled-DAG 3 and 4 of OTA3 have the same architectures, which composes of one mix convolution, one atrous convolution, and one normal convolution layer. This phenomenon shows that this sub-graph structure could be a good fit to be constructed in the final stage to detect the offset voltage metric. To summary, our search strategy produces a significantly more complex model for OTA1, helping extract placement features on various scales to boost the performance of this large design model.

We also provide a summary of these NAS-crafted models for four OTA designs in Table 5 by presenting the number of vertices and edges in sampled-DAGs at each sampling layer. This table indicates models for large design (OTA1 and OTA2) indeed need more operations than models for small designs to predict the layout performance accurately. Also, OTA2 has much more vertices and edges than OTA1. Thus, different transistor sizing also largely affects the model size and edge connections

Table 4: Validation of the customization effectiveness. We take OTA1 data and perform training on customized models of other designs from scratch.

Training Data	Model Development	Accuracy ↑
Training on OTA1	Searched for OTA1	**91.39**
	Searched for OTA2	90.59
	Searched for OTA3	91.24
	Searched for OTA4	91.24
Training on OTA3	Searched for OTA1	92.31
	Searched for OTA2	90.99
	Searched for OTA3	**93.57**
	Searched for OTA4	91.61

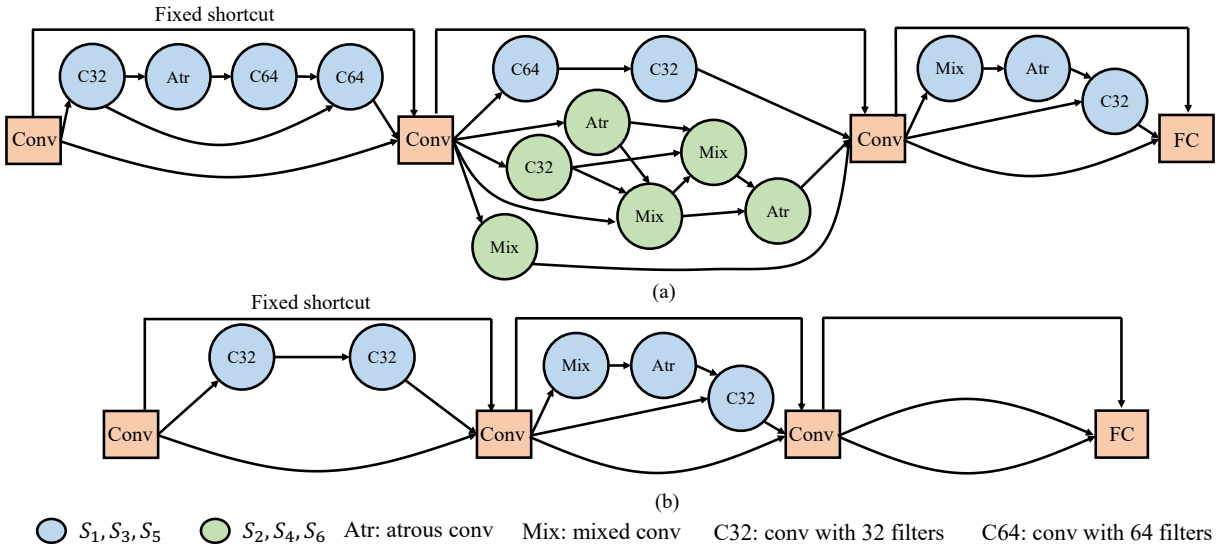

Figure 5: NAS-crafted models (a) for OTA1 and (b) for OTA3.

Table 5: Summary of our NAS-crafted models.

Design	sampled-DAG 1, 2		sampled-DAG 3, 4		sampled-DAG 5, 6	
	#vertex	#edge	#vertex	#edge	#vertex	#edge
OTA1	4	7	8	16	3	6
OTA2	6	14	6	11	8	14
OTA3	2	4	3	6	0	2
OTA4	8	10	3	5	4	6

because OTA1 and OTA2 have the same schematic but processed with different sizing.

In addition to transistor sizing, different target layout quality metrics also have impact on models. The comparison between OTA3 and OTA4 in Table 5 (similar #transistors but different metrics) indicates that predicting CMRR requires more computations than offset voltage. In summary, with different transistor numbers, transistor sizing, and layout quality metrics, we need to design and craft suitable model structures to provide better prediction performance. This analysis provides some insights into future analog placement quality prediction model development.

7 CONCLUSION

This work proposes a generalized framework, which can be applied in both digital and analog design automation, to automate the development of placement quality estimators without any human interference. This is the first work focusing on automated model development for analog designs. Under the framework, we can provide customized ML models for different analog circuit designs and different performance metrics. These customized models prove to outperform previous layout performance estimator. We also demonstrate the effectiveness of our search strategy and customization method with random search and extra experiments.

In addition, our search strategy exhibits high efficiency and scalability. The search process only takes 0.5 days to develop high-performance models, shortening the model development cycle in real-world applications. Based on automatically generated models, we provide insights to facilitate the future analog performance models development. With different design characteristics and target layout quality, we need to craft different model structures to maximize prediction performance. In the future, we plan to apply our generalized framework to a large range of essential ML for EDA problems in both digital analog layout automation.

ACKNOWLEDGMENTS

This work is supported by SRC GRC-CADT 3103.001/3104.001, NSF CCF-2106725/2106828, and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

REFERENCES

[1] Han Cai, Ligeng Zhu, and Song Han. 2018. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332* (2018).
[2] Chen-Chia Chang et al. 2021. Automatic Routability Predictor Development Using Neural Architecture Search. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*.
[3] Liang-Chieh Chen et al. 2017. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587* (2017).
[4] Hsin-Pai Cheng et al. 2019. SwiftNet: Using Graph Propagation as Meta-knowledge to Search Highly Representative Neural Architectures. *arXiv preprint arXiv:1906.08305* (2019).
[5] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[6] Kishor Kunal et al. 2019. ALIGN: Open-source analog layout automation from the ground up. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–4.
[7] Koen Lampaert et al. 1995. A performance-driven placement tool for analog integrated circuits. *IEEE Journal of solid-state circuits* 30, 7 (1995), 773–780.
[8] Yaguang Li et al. 2020. A customized graph neural network model for guiding analog IC placement. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.
[9] Mark Po-Hung Lin et al. 2016. Recent research development and new challenges in analog layout synthesis. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 617–622.
[10] Po-Hung Lin et al. 2009. Analog placement based on symmetry-island formulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 6 (2009), 791–804.
[11] Mingjie Liu et al. 2020. Towards decrypting the art of analog layout: Placement quality prediction via transfer learning. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 496–501.
[12] Qiang Ma et al. 2010. Simultaneous handling of symmetry, common centroid, and general placement constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 1 (2010), 85–95.
[13] Mingxing Tan and Quoc V Le. 2019. Mixconv: Mixed depthwise convolutional kernels. *arXiv preprint arXiv:1907.09595* (2019).
[14] Po-Hsun Wu et al. 2012. Performance-driven analog placement considering monotonic current paths. In *Proceedings of the International Conference on Computer-Aided Design*. 613–619.
[15] Linfu Xiao et al. 2010. Practical placement and routing techniques for analog circuit designs. In *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 675–679.
[16] Biying Xu et al. 2019. Magical: Toward fully automated analog ic layout leveraging human and machine intelligence. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.
[17] Biying Xu, Shaolan Li, Chak-Wa Pui, Derong Liu, Linxiao Shen, Yibo Lin, Nan Sun, and David Z Pan. 2019. Device layer-aware analytical placement for analog circuits. In *Proceedings of the 2019 International Symposium on Physical Design*. 19–26.
[18] Keren Zhu et al. 2019. Geniusroute: A new analog routing paradigm using generative neural network guidance. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.