Toward Fully Automated Machine Learning for Routability Estimator Development

Chen-Chia Chang, Jingyu Pan, Zhiyao Xie, *Member, IEEE*, Tunhou Zhang, *Graduate Student Member, IEEE*, Jiang Hu, and Yiran Chen, *Fellow, IEEE*

***Abstract*—The rise of machine learning (ML) technology inspires a boom of its applications in electronic design automation (EDA) and helps improve the degree of automation in chip designs. However, manually crafting ML models remains a complex and time-consuming process because it requires extensive human expertise and tremendous engineering efforts to carefully extract features and design model architectures. In this work, we leverage automated ML techniques to automate the ML model development for routability prediction, a well-established technique that can help to guide cell placement toward routable solutions. We present an automated feature selection method to identify suitable features for model inputs. We develop a neural architecture search method to search for high-quality neural architectures without human interference. Our search method supports various operations and highly flexible connections, leading to architectures significantly different from all previous human-crafted models. Our experimental results demonstrate that our automatically generated models clearly outperform multiple representative manually crafted solutions with a superior 9.9% improvement. Moreover, compared with human-crafted models, which easily take weeks or months to develop, our efficient automated machine learning framework completes the whole model development process with only 1 day.**

***Index Terms*—Automated machine learning (AutoML), neural architecture search, physical design.**

I. Introduction

MODERN digital integrate circuit (IC) design consists of many complicated stages that cannot be decoupled with each other. Although existing electronic design automation (EDA) methods have demonstrated their effectiveness, the current framework lacks of reliable prediction of the quality of outputs from early stages. This limitation implies that solutions produced by early steps may yield unsatisfactory solution qualities in later steps. Thus, designers need to spend many design iterations to achieve an optimal quality, which significantly prolongs the overall turnaround time and development duration.

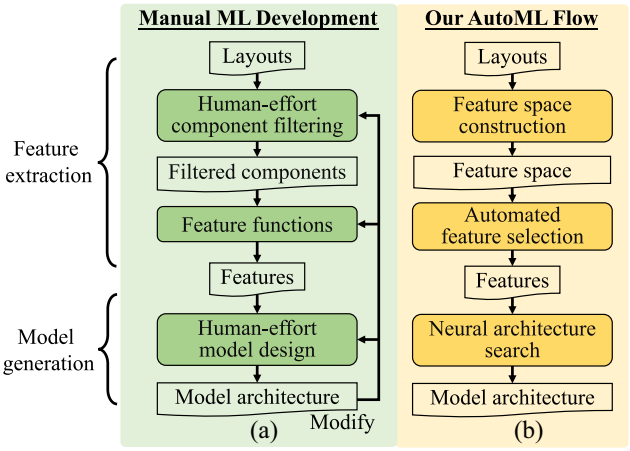


Fig. 1. (a) Manual ML model development and (b) our AutoML flow for EDA.

Machine learning (ML) techniques have been massively adopted to facilitate the interactions between design steps by enabling early stage predictions [1], [2], [3], [4], [5], [6]. For example, ML models are applied to predict whether decisions at early design steps will lead to satisfactory design objectives in subsequent steps. With fast feedback from ML models, a design can be converged to a high-quality solution with significantly fewer iterations than traditional EDA flows. As a result, the overall turnaround time can be significantly reduced, leading to a more efficient design process. Deep learning (DL) techniques, a subcategory of ML, are also widely applied to increase the predictability between different design stages for modern circuits. In existing works, convolutional neural network (CNN) models [3], [5], [7] and generative adversarial network (GAN) models [4], [8] are popular choices of the applied DL models.

ML model development for EDA mainly involves two steps, feature extraction and model generation, as shown in Fig. 1. For feature extraction, the designer manually designs and selects features that are expected to benefit the prediction of the circuit/layout. The selected feature set usually needs to be modified several times to remove irrelevant ones to prevent model overfitting. Then, for model generation, the designer hand-crafts a model architecture that could perform well on the EDA task. This process also requires multiple trials to

Manuscript received 20 March 2023; revised 20 July 2023 and 21 October 2023; accepted 25 October 2023. Date of publication 7 November 2023; date of current version 21 February 2024. This work was supported in part by SRC GRC-CADT under Grant 3103.001/3104.001, and in part by NSF under Grant CCF-2106725/2106828. This article was recommended by Associate Editor D. Z. Pan. *(Corresponding author: Chen-Chia Chang.)*

Chen-Chia Chang, Jingyu Pan, Tunhou Zhang, and Yiran Chen are with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: chenchia.chang@duke.edu; jingyu.pan@duke.edu; tunhou.zhang@duke.edu; yiran.chen@duke.edu).

Zhiyao Xie is with the Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Hong Kong, SAR (e-mail: eezhiyao@ust.hk).

Jiang Hu is with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: jianghu@tamu.edu).

Digital Object Identifier 10.1109/TCAD.2023.3330818

guarantee optimal performance. In the works [1], [2], [3], [4], [5], [6] of ML for EDA, researchers manually select and extract circuit-related components and features, e.g., the cell-density map, that are correlated to the prediction objectives. Also, they all propose different hand-crafted model architectures for training. An experienced developer may easily take months to complete selecting features and designing a neural network architecture for such EDA applications. Moreover, the growing popularity of DL techniques in EDA has led to an increased complexity of the model development because DL models are able to handle a large number of features and model layers that can be constructed with various operations. Thus, both feature selection and architecture design require extensive expertise on both ML and EDA and tremendous engineering efforts. This challenge significantly prolongs the development cycle of ML-based EDA tools and exacerbates development cost.

Automated machine learning (AutoML) [9] enables design automation of ML model development, including automated feature selection and neural architecture search (NAS), without (or with minimum) human interventions. Based on the raw feature space, automated feature selection efficiently constructs an effective feature subset to form the dataset. Based on the target dataset, NAS identifies an architecture search space and then applies certain search strategies, such as reinforcement-learning-based [10] methods or evolutionary-guided [11] methods, to judiciously discover promising architectures. Additionally, neural network architectures provided by NAS [12] have shown to outperform state-of-the-art manual designs with significantly improved model accuracy in other domain applications, e.g., computer vision. AutoML techniques have demonstrated its ability to generate high-quality ML models without burdensome manual development process. As some ML applications for EDA [3], [4], [6] can be represented and processed like computer vision tasks, it is natural to leverage AutoML techniques to automate the ML model development for EDA.

In this work, we propose an AutoML framework, including an automated feature selection process and an NAS method, to reduce the time and effort of developing ML models for EDA while improving the model performance. The flow of our AutoML framework is sketched in Fig. 1. Our automated feature selection process provides an intelligent manner to efficiently select useful raw features for model inputs. In addition, our NAS search space is abstracted as graphs that enable rich and flexible feature interactions to better capture congestion patterns. We can identify high-quality models by adopting graph propagation as our search strategy. Therefore, we can efficiently obtain ML models that deliver superior performance. To demonstrate the concept, we apply our AutoML framework on a well-known research topic, routability prediction [3], [4], [6]. Routability prediction estimates the routability of design solutions at the placement stage [13] with the following application scenario: design rule checking (DRC) hotspot detection. DRC hotspot detection identifies the locations of design rule violations (DRVs) after detailed routing stage. This routability application can be used to guide DRC violation mitigation techniques. In this case study, our goal is to efficiently and effectively select features and design model architectures for DRC hotspot detection. In the mean time, our framework can provide a standard raw feature set and some high-performing model architectures to facilitate routability prediction development.

Our main contributions are summarized as follows.

1) We propose an AutoML framework that consists of automated feature selection and NAS to develop routability estimators with minimum human interference. To the best of our knowledge, this is the first exploration of a comprehensive AutoML framework for EDA.
2) We propose an automated ML-based feature selection method that efficiently and effectively selects features to improve the model accuracy and the search efficiency of our NAS method.
3) We develop a graph-based search space that supports various operations and flexible connections and a graph propagation search strategy to guide the search process toward high-quality architectures.
4) The model crafted by our AutoML framework outperforms two representative routablity estimators [3], [6] with a superior 9.9% higher area under the receiver operating characteristic (ROC-AUC). The entire AutoML flow can be completed in 1 day, demonstrating high efficiency of our framework.
5) We provide a thorough analysis of our search outputs, i.e., the extracted features and NAS-crafted models, to provide insights for the future routablity estimator development.

Through this article, we demonstrate the potential of AutoML techniques to facilitate the effectiveness and efficiency of ML development for EDA. In addition, our framework is open-sourced to encourage future studies in this area.

The remainder of this article is organized as follows. Section II details the preliminaries, including the definition of DRC hotspot, the prior exploration for routability prediction, and the concepts of automated feature extraction and NAS. Section III formulates our AutoML problem for routability prediction. Section IV introduces our AutoML framework with two main components: 1) automated feature extraction in Section IV-A and 2) NAS in Section IV-B. Section V presents the experimental results. Section VI gives the analysis of our AutoML-crafted features and models. Finally, Section VII concludes this article.

II. Preliminaries

In this section, we first give a brief introduction to the DRC hotspot, a metric used to evaluate the routability of a routing result. Then, we discuss the current research progress on routability prediction, including features and model architectures, used in the prior literatures. Finally, we introduce the concepts of automatic feature selection and NAS, which are the two key stages in our AutoML framework.

A. DRC Hotspot

DRC hotspot is a 2-D map that pinpoints DRV positions of the detailed routing result. Thus, DRC hotspot directly indicates the regions that truly need to be refined after routing. Fig. 2 depicts the examples of DRVs. In Fig. 2(a), a short

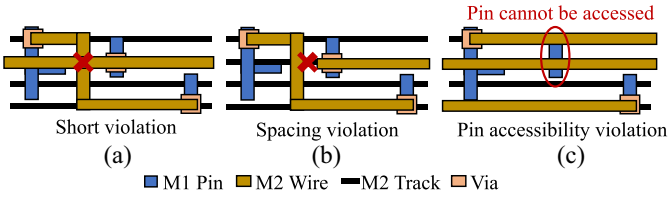


Fig. 2. (a) Short violation: Two wires in M2 are overlapped. (b) Spacing violation: Two wires are too close. (c) Pin Accessibility Violation: The M1 pin cannot be accessed by any M2 wires.

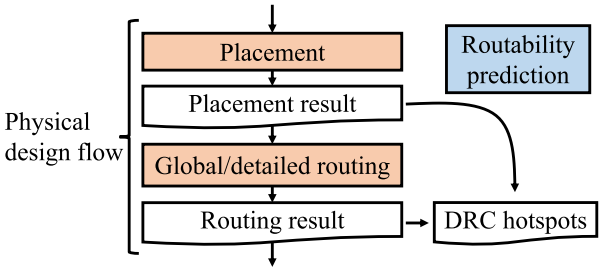


Fig. 3. Physical design flow and routability prediction. After the placement, the placement result comes into the global and detailed routing stages to generate the routing result. With the ML model, we can directly use the placement result to predict DRC hotspots.

violation occurs when two wires are overlapped, which will cause the circuit to malfunction. In Fig. 2(b), a spacing violation happens if the distance between two wires is smaller than a threshold, which will cause crosstalk. In Fig. 2(c), a pin accessibility violation arises when the upper layer of a pin is covered by other wires, and thus the net cannot connect to the pin.

B. Machine Learning for Routability Prediction

To evaluate routability of a placement solutions, traditionally we need to come through the global routing and the detailed routing to get DRC hotspots. However, routing is a very time-consuming process that prevents the placer from efficiently examining the routability. To overcome this challenge, researchers have developed early routability prediction techniques that enables designers or EDA tools to perform preventive measures such that DRC hotspot can be avoided in a proactive manner. Fig. 3 illustrates the routability prediction flow, which takes the placement result to forecast DRC hotspot. This is a representative topic in ML for EDA, and its benefits to chip quality are well demonstrated in many previous works [3], [5], [6], [7], [14], [15]. In this section, we introduce the ML background about routability prediction.

Feature Extraction: Feature extraction for routability prediction aims to select useful information from the placement results into a format compatible to ML models. All previous exploration uses the cell and net-related features to facilitate routability prediction. The works [14], [15] extract features into numerical format, and the others [3], [5], [6], [7] employ image-based features to perform the prediction. However, all these features are still manually selected, requiring considerable engineering effort to choose important features and discard the redundant ones. Also, these hand-crafted features may lead to suboptimal solutions because all previous works typically give specified features in a predefined manner, without offering a complete feature selection or analysis process. Thus, an efficient and effective automated feature selection approach is necessary to accelerate the selection process and ensure the quality of the selected features. A work [16] develops a model architecture with lattice graph to mimic the cell/net feature extraction. However, it still needs to manually design model architectures and decide their target features. Also, their goal is different from our framework, as our focus lies in the automated selection of features for the model input. To the best of knowledge, there is no prior work focusing on the automated feature selection on ML for EDA.

Model Architecture: Both works [14], [15] chooses multivariate adaptive regression splines to forecast the detailed routing routability. In recent years, deep neural network methods, including CNN and fully convolutional network (FCN), become the dominant solutions [3], [4], [5], [6], [7] to routability prediction. As routability prediction requires pinpointing specific locations with DRC hotspots in a 2-D layout, it shares a similar setting with semantic segmentation in identifying pixelwise properties. Thus, FCN, as a popular technique for semantic segmentation, is widely used in routability prediction [3], [5], [6].

Among representative routability estimators in recent years, RouteNet [3] and J-Net [5] propose U-Net [17]-like FCN structures, PROS [6] adopts an encoder–decoder FCN framework, and the work [4] proposes a conditional GAN [4] (cGAN)-based method with FCN architecture for routability prediction. To the best of our knowledge, all previous routability estimators [3], [5], [6], [7], [14], [15] are designed by human developers. Thus, they require both ML and EDA expertise and easily take weeks of model development time. In addition, previous works develop their models mostly based on hierarchical structures, with a limited number of branch structures. In comparison, our graph-based search space enables highly flexible connections and rich branches, thus providing significantly different model structures. The details of our search space are presented in Section IV-B. Since the complex pattern behind routability prediction may be reflected by complicated interactions among features in a wide layout region, branch structures can capture combined information from different sources and benefit model performance.

C. Automated Feature Selection

Automated feature selection enables the automation of feature selection given the raw feature space without human intervention. In detail, it aims to efficiently establish a suitable feature subset to maximize the model performance because redundant features in the feature space could cause overfitting, leading to suboptimal model performance. Additionally, reducing the feature number can accelerate the training process, which can significantly enhance the NAS efficiency because NAS often needs lots of training steps to explore architectures. Thus, a succinct yet effective feature set is required for the AutoML flow.

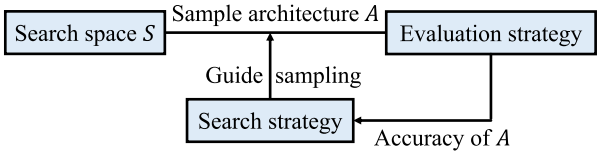


Fig. 4. Overview of NAS.

In other domain applications, there has been some approaches [9] for feature selection, e.g., iterative and evolution-based search algorithms. However, iterative selection methods are inefficient when we have a large feature combination. Also, evolution-based search algorithms require large amounts of model training periods to converge. Given the poor scalability of previous methods, we design a novel ML-based feature selection method which can complete selection in an efficient manner. Through our approach, we can identify the most useful feature combinations for DRC hotspot detection.

D. Neural Architecture Search

NAS [10] automatically conducts architecture engineering to find effective neural network models for specific tasks. Recent works demonstrate great potential of NAS in applications, including image classification [10], object detection [10], and semantic segmentation [18]. NAS contains three key ingredients: 1) *search space*; 2) *evaluation strategy*; and 3) *search strategy*. Search space defines a family of candidate architectures that can be explored in NAS. Evaluation strategy determines the way 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 promising ML models. The overall NAS procedure is sketched in Fig. 4.

Our NAS approach abstracts NAS space into graphs and includes various types of operations as search options. Then, our NAS approach performs proxyless evaluation [19] to evaluate the performance of candidate models on the target dataset. Finally, our NAS approach develops a search strategy based on a progressively graph updating and sampling algorithm [20] to efficiently explore ML models.

III. Problem Formulation

We apply AutoML techniques to assist the feature extraction and the design of ML models for routability prediction. After placement, a layout is tessellated into $w \times h$ tiles, then its input feature $X \in \mathbb{R}^{w \times h \times c}$ is composed of c different 2-D feature maps. The ground-truth label is collected after detailed routing finishes for DRC hotspot. Our AutoML problem can be formulated as follows.

Problem 1 (AutoML for Routability Prediction): Given a set of placement solutions, it aims to select the input features X from the feature space C, and then explore the architecture $A \in S$ of the neural network model f_A within the defined search space S, to detect the locations of DRC hotspots Y such that the performance of f_{A_L} is maximized, where

$$f_A : X \in \mathbb{R}^{w \times h \times c} \rightarrow Y \in \{0, 1\}^{w \times h}.$$

We use the ROC-AUC as the metrics of diagnostic ability of the model. ROC curve plots the tradeoff between true positive rate (TPR) versus false positive rate (FPR) by varying classification threshold. A higher ROC-AUC indicates that a higher precision of DRC hotspot detection can be achieved at the cost of the same number of false alarms.

IV. Automated ML Model Development

Our AutoML framework consists of two main stages: 1) automated feature selection that extracts the most relevant features within the feature space and 2) NAS which designs the high-quality model architecture based on the extracted features. In this section, we first present our automated feature selection method. Then, we introduce our NAS approach, including the search space, the evaluation strategy, and the search strategy.

A. Automated Feature Selection

Automated feature selection aims to efficiently and effectively choose useful features to boost the model performance and the search efficiency of our NAS method. To achieve this goal, we establish a comprehensive raw feature space that covers common raw features relevant to routability prediction. Since defining the feature space still requires human expertise, we include widely known raw features that could correlate with routing results based on the conventional understanding of placement and routing practices from previous routability prediction works [3], [4], [6], [21] to minimize the human effort. These features can be separated into two categories: 1) cell density-based features and 2) net density-based features. We first introduce the cell density-based features as follows.

1) *Macro Density:* The region occupied by macros.
2) *Cell Density:* The number of standard cells in a region.
3) *Pin Density:* The number of pins in a region.
4) *Pin Accessibility:* The potential number of wires connecting to the pins in each region [6].

Note that cells with different functions and nets with different bounding box sizes would have distinct impacts on routability. Thus, density distributions of the D flip-flips (DFFs) and clock tree buffers are generated independently for the cell, pin, and pin accessibility features. In addition, four pin density features is derived by the pins associated with nets that have varying bounding box thresholds (4k, 1k, 0.1k, 0.01k μm^2). In summary, we have 14 cell density-based features.

Next, the *net density-based features* are provided below.

1) *RUDY:* The total uniform wire volume spreading in bounding boxes of nets [3].
2) *Bounding Boxes:* The total number of the net bounding box outlines passing each region [21].
3) *Vertical/Horizontal Net Density:* The uniform wire volume in the vertical/horizontal direction [6].
4) *Rectangle Steiner Minimum Tree (RSMT) RUDY:* The total uniform wire volume spreading in the RSMTs of nets.
5) *Pin RSMT RUDY:* This feature assigns the density of each pin to the RSMT RUDY of the associated net [3].

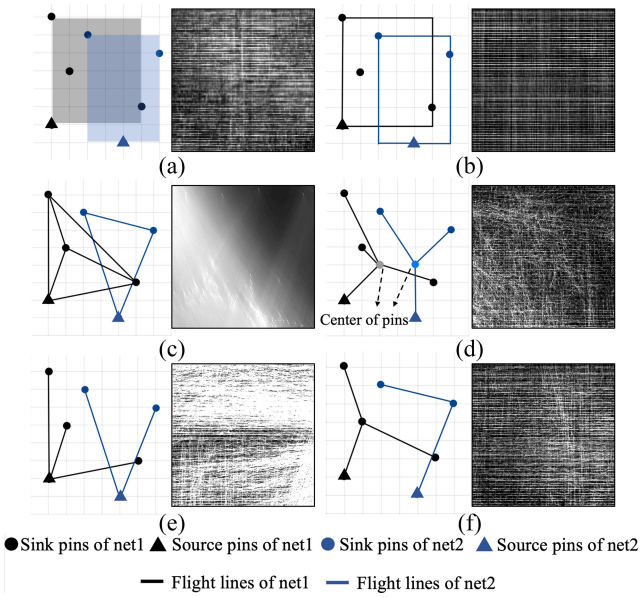


Fig. 5. Visualization of wire density features. (a) RUDY. (b) Net bounding boxes. (c) Pairwise flight lines. (d) Star flight lines. (e) Source-sink flight lines. (f) MST flight lines.

To estimate the net connectivity, a heuristic named flight line [21], which is a line that connects two pins, is adopted.

1) *Pairwise Flight Lines:* Connect from every pair of pins in each net.
2) *Star Flight Lines:* Connect every pin to the centroid of all pins in each net.
3) *Source-Sink Flight Lines:* Connect the source pin to all sinks in each net because routers tend to connect sinks to the source through shortest paths in timing optimization.
4) *Minimum Spanning Tree (MST) Flight Lines:* Connect the pins through MST in each net.

For each type of flight line, we draw flight lines of all nets into a single image to construct the feature. Some net density-based features are visualized in Fig. 5.

In addition to constructing all 10 net density-based features for all nets, we also separate nets into four groups with bounding box thresholds (4k, 1k, 0.1k, 0.01k μm^2) to construct four features for each wire density. As a result, there are 50 net density-based features. In summary, cell density-based features and net density-based features together form a comprehensive feature space $C \in \mathbb{R}^{w \times h \times 64}$ to cover all potential raw features for routability prediction.

Training with the entire feature space could lead to suboptimal model performance because some uncorrelated features may introduce redundancy or noise. In addition, training with large amount of features significantly prolongs the training period, which can severely worsen the efficiency of our NAS process. Thus, we propose an ML-based automated feature selection method to capture the critical features from the feature space and build a succinct feature set.

First, we allocate the weight $w \in \mathbb{R}^{h,h,c,o}$ of the first convolutional layer into c different groups W_i according to the input channel dimension

$$W_i = \{w_{j,k,i,l} | \forall j, k < h \quad \forall l < o\}$$

where h is the kernel size, c is the number of input feature channels, and o is the number of output channels. Thus, each group W_i of weight will correspond to an input feature i. Then, we employ a structured regularization loss called group-lasso ℓ_g [22] on every W_i

$$L_g = \sum_i \ell_2(W_i) \tag{1}$$

where $\ell_2(W_i)$ is the mean square root of weights in W_i. Namely, this group-lasso loss applies ℓ_1 loss on $\ell_2(W_c)$. Our overall training objective is

$$\min L_p + \alpha * L_g$$

where L_p is the prediction loss for routability prediction, and α is the regularization strength of the group lasso. Minimizing L_g during training encourages weights W_i between different groups to become sparse, leading to some groups having weights close to zero. In addition, minimizing L_p together can help us maintaining high-prediction performance at the same time. Note that each W_i is responsible for a specific input feature i. Therefore, we can view the norm of W_i as the feature importance metric. The higher value of the norm of W_i means that this feature contributes more to the routability prediction result. Then, we sort features based on W_i in a nonincreasing order and iteratively add the feature C_i into our feature subset $X_{i+1} = X_i \cup C_i$ for $i = 1$ to $|C|$ and perform training. Finally, we select the best-performing feature set X_i as our input feature set X.

Our ML-based feature selection flow is illustrated in Fig. 6. Since we use a graph-based search space in our NAS (will be detailed in Section IV-B), we apply the model with six complete-ordered directed acyclic graphs (DAGs), which covers all possible architectures in our search space, in our feature selection.

Also, our ML-based feature selection method only requires $|C|$ times of training to complete selection, making it much more efficient compared to other iterative or evolution-based search algorithms. In addition, we empirically observe that the feature selection results are not affected by the choice of model structures used in group lasso training. Fig. 7 shows the high similarity between the norms of group weights trained with two different model architectures. With this special attribute of our feature selection method, it is suitable to design a two-stage AutoML framework.

B. Graph-Based Neural Architecture Search

Based on the selected input features, we use our proposed graph-based NAS method motivated by [20] to automate the design of neural networks for routability prediction. In the following, we introduce three key components of our NAS: 1) search space; 2) evaluation strategy; and 3) search strategy.

Search Space: In our NAS-based model, we can partition the architecture into two parts: one part is iteratively changed during the search process, while the other is fixed. Our model is shown in Fig. 8. The yellow rectangles represent the fixed part with widely adopted structures, and the six blue rectangles indicate the changeable part. In the following, we demonstrate the architectures of the fixed part and the changeable part.

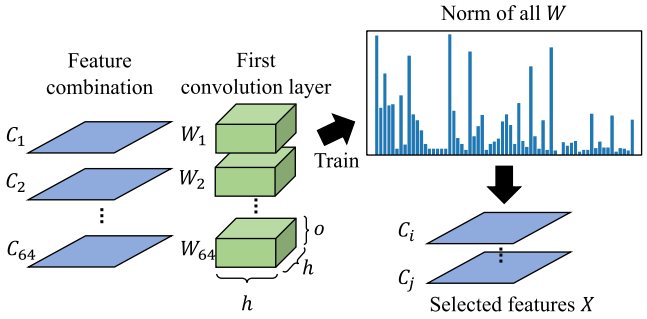


Fig. 6. Our ML-based automated feature selection flow. We group the weights of the first convolution layer according to input features. After training with the group lasso method, we obtain the norm of each group. Based on these norms, we iteratively add the top-score features into the feature set and perform training to select the best-feature subset.

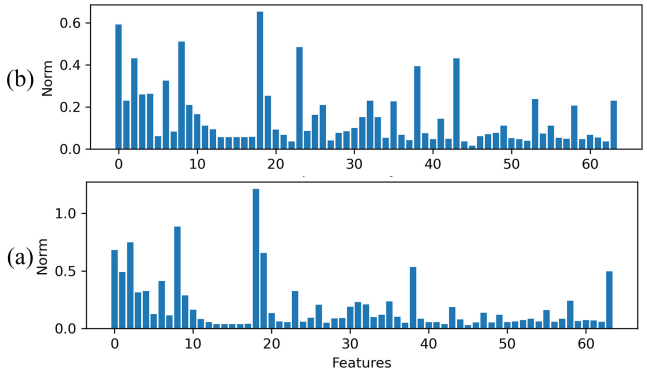


Fig. 7. These plots display the group norms of weights that were trained using the (a) RouteNet [3] structure and (b) the architecture comprising of six fully ordered DAGs based on our NAS.

We start with deciding all candidate operations in the changeable part. First, regular convolution layers with different numbers of filters are included. Besides, atrous convolution, also named dilated convolution [23], is selected as a promising candidate operation since it can effectively enlarge receptive fields of filters. This operation can thus help to capture large patterns, such as congestions, caused by nets spanning a large region. In addition, the work of [24] introduces a new mixed depthwise convolution (MixConv) that separates channels into groups and applies different kernel sizes to each group. Compared with a regular convolution that can only observe patterns in a fixed size area, this operation can identify congestion patterns of different sizes when applied in routability prediction. Thus, MixConv is a good fit for our work since routability can be affected by the relations of nets and standard cells in different regions within a layout. Among these candidate operations, to our best knowledge, atrous convolution is only adopted in a recent routability estimator [6], and MixConv is never used in routability prediction. Adopting various promising operations can improve diversity in candidate models and help cover more potential high-quality models in the search space. As a result, the candidate operations *op* include the following four types.

1) 3×3 convolution with $64 * 2^i$ filters.
2) 3×3 convolution with $128 * 2^i$ filters.
3) 3×3 atrous convolution with dilation rate 2, $64 * 2^i$ filters.
4) Mixed convolution with four groups, kernel size [7, 9, 11, 13].

Note that our search space totally has three stages, and we use a multiplier of 2^i to gradually increase the number of filters when the operation is selected in stage i.

By viewing CNN/FCN as a set of operations and the connections of operations, a model can be regarded as a graph. Specifically, vertices represent operations and edges indicate the directed connections of operations. Therefore, we view the six blue changeable parts $\{S_1, S_2, \ldots, S_6\}$ in Fig. 8 as DAGs, named sampled-DAGs. There are two parallel sampled-DAGs between every two downsampling layers. The changes of them are restricted and guided by six guide-DAGs $\{G_1, G_2, \ldots, G_6\}$. Each guide-DAG $G_i(V_i, E_i)$ represents a combination of the candidate operations and the propagation of data tensors. It is composed by a set of completely ordered vertices V_i, with each vertex $v \in V_i$ representing a candidate operation op_v. Each edge $e(u, v) \in E_i$ represents the propagation of the output tensor of vertex u to the input of v. Edge $e(u, v)$ is constructed if $u < v$ in their order, which makes the guide-DAG G_i completely ordered with maximum edges to provide all possible connections. Fig. 9(a) shows an example of a guide-DAG, where the complete order of vertices is $1 \rightarrow 2 \rightarrow \cdots \rightarrow 7$. Each vertex concatenates all its input tensors from the incoming edges and produces the output tensor by its operation. Specifically, given a vertex v with op_v and input tensors $i_{u_1}, i_{u_2}, \ldots, i_{u_k}$ from incoming edges $e(u_1, v), e(u_2, v), \ldots, e(u_k, v)$, the output tensor o_v of this vertex is

$$o_v = op_v\big(\text{Concat}(i_{u_1}, i_{u_2}, \ldots, i_{u_k})\big).$$

Concatenation with all the input tensors of each vertex can help the model to discover different feature combinations to enhance the observation of routability information.

To connect each sampled-DAG together to form our sampled model, we design some fixed operations in the fixed part. As Fig. 8 shows, there are three downsampling layers with standard convolution with stride of 2. The fixed structure at the end is composed of three transposed convolution layers with stride of 2 to recover feature maps with a total upsampling factor of 8 to form a DRC hotspot solution. In addition, we add three shortcuts between stages with different feature scales to further boost the performance, following the famous U-Net [17] model.

In our search space, the parallel sampled-DAG structures between every two downsampling layers can produce different feature representations and pass their aggregation to the next sampled-DAG structures after downsampling. Additionally, our search space allows for multiple parallel tensor propagations within a sampled-DAG because the topology of the graph G_i contains all possible connections. In summary, our approach offers a highly flexible and extensive search space that can generate a variety of feature representations and lead to accurate routability prediction.

Evaluation Strategy: Previous work [19] proposes conducting NAS directly on the target dataset to enhance the performance of searched models. Therefore, we directly perform search on the training split of our target dataset,

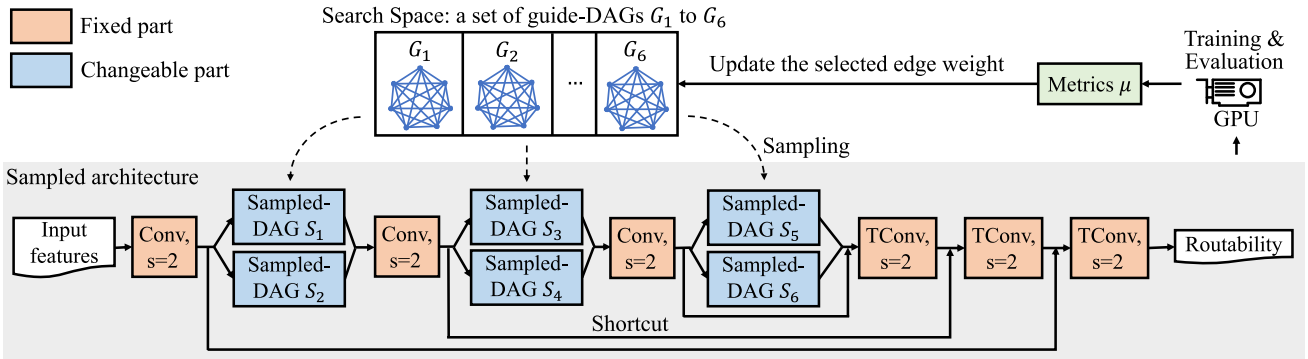


Fig. 8. Overview of our graph-based NAS method. We sample multiple DAGs from the search space to form a sampled model. After training for several epochs, we get the evaluation metrics, i.e., ROC-AUC, and utilize it to update the weight of selected edges in guide-DAGs. Conv, $s = 2$ and TConv, $s = 2$ represent standard convolution, and transposed convolution layer with stride 2, respectively.

Algorithm 1 Connection Weights Update in the Meta Graph

Require: A set of guide-DAGs $\{G_1, \ldots, G_6\}$, baseline metrics β, learning rate α

1: **for** $i = 1$ to 6 **do**
2: $G'_i = \text{Preprocess}(G_i)$ ▷ pseudo vertex v_0 insertion
3: **while** η not converge **do**
4: **for** $i = 1$ to 6 **do**
5: $S_i = \text{Sampling}(G'_i)$ ▷ Algorithm 2
6: $M = \text{ConstructModel}(\{S_i, i = 1 \text{ to } 6\})$
7: $\eta = \text{Eval}(M)$
8: **for** $i = 1$ to 6 **do**
9: **for** edge e in G'_i **do**
10: **if** e is selected in S_i **then**
11: $w_e = w_e * \exp(\alpha(\eta - \beta))$
12: $\beta =$ average metrics of top-5 sampled graphs

employing ROC-AUC as the search objective for routability prediction (mentioned in Section III).

Search Strategy: Given the large size of our search space, exhaustively examining every subgraph in the search space is neither efficient nor feasible. For example, it is impossible to use random search to efficiently converge and obtain high-quality model architectures. To overcome this challenge, we propose a *graph propagation* method to search for well-performing models in an efficient manner.

Graph propagation has two key components to perform searching: 1) graph sampling and 2) weight updating techniques. Graph sampling is to sample edges and operations from the guide-DAG G_i by defining a weight on each edge to control its sampling probability. In weight updating, we will gradually update weights through our search process to find a promising model within the search space. The flow of our search strategy is detailed in Algorithm 1. First, in the preprocessing step (lines 1 and 2), we construct a pseudo vertex v_0 and add v_0 to V_i. Vertex v_0 represents the downsampling layer before G_i. An edge $e(v_0, v)$ is constructed for each $v \in V_i$. These edges provide all possible input connections from the downsampling layer v_0 to all vertices $v \in V_i$. The graph after preprocessing is denoted by $G'_i(V'_i, E'_i)$. The edge weights in E'_i are set to 1. An example of G'_i is shown in Fig. 9(b), where the new green vertex with index 0 is a pseudo vertex, representing the downsampling layer before G_i in Fig. 9(a).

After preprocessing, we enter the iterations to optimize our model by its performance (line 3 of Algorithm 1). In each iteration, we apply the sampling function in Algorithm 2 on each G'_i to sample the corresponding S_i (lines 4 and 5). In the remaining paragraph, we will cover the Algorithm 2, which takes G_i as the input and outputs a sampled-DAG $S_i(V_{S_i}, E_{S_i})$. First, we initialize V_{S_i} with v_0 (line 1), the vertex that represents the downsampling layer. For each vertex $v_j \in V'_i$, if v_j is in V_{S_i}, we iterate through all its edges $e(v_j, v_k)$ to perform the edge selection (lines 3–5). During the edge selection, for each $e(v_j, v_k)$, its edge selection probability p is set to the normalized weight of $w_{e(v_j,v_k)}$ over the weights of all outgoing edges of v_j. This normalization is performed with a softmax function (line 6). Note that a larger weight edge means a higher probability to be sampled, and the softmax function can further enhance the difference between weights and reflect it on the probability. If $e(v_j, v_k)$ is selected, we add $e(v_j, v_k)$ and v_k into E_{S_i} and V_{S_i}, respectively, (lines 8–10). In later iterations, the outgoing edges of v_k will be extracted and performed sampling. After the edge selection of S_i, we enter the operation sampling (lines 11–19) to select the operation of each vertex $\in V_{S_i}$. Each vertex v_j also has a weight $wop_{k v_j}$ to represent the sampling probability of four different operation k. Similar to edge sampling, we use softmax function to normalize operation weights and then select one operation for each vertex. Finally, S_i is returned after iterating through all the vertices. Fig. 9 demonstrates an example of subgraph sampling. In Fig. 9(b), red edges in G_1 are selected by Algorithm 2 in edge sampling. According to the red edges, the sampled-DAG S_1 is constructed in Fig. 9(c). Vertices without any outgoing edge are connected to the right downsampling layer. Then, we perform operation sampling on S_1 to select vertex operations to form the sampled architecture in Fig. 9(d).

After sampling each S_i, we construct our model through the architecture in Fig. 8 and measure our evaluation metrics η with our evaluation strategy in Algorithm 1 (lines 6 and 7). According to η, we iterate all edges in each G_i and update the edge weights which are sampled in S_i in this iteration

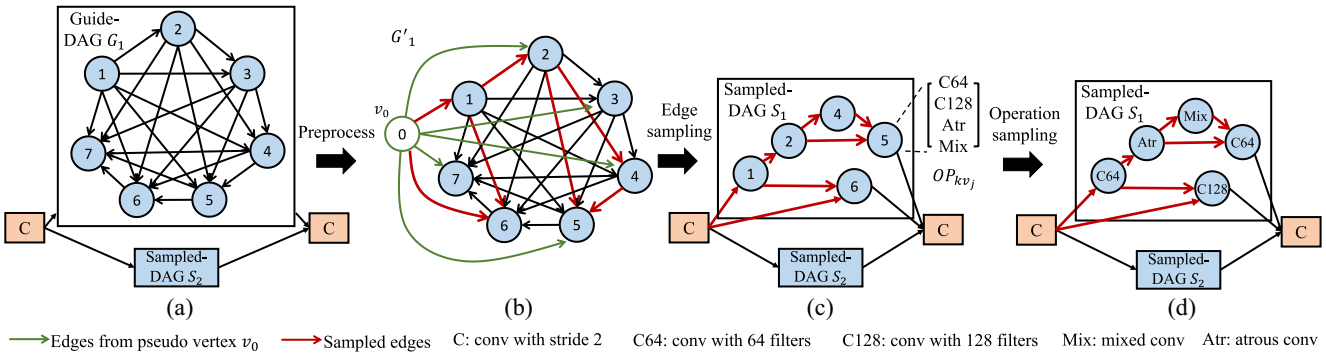


Fig. 9. Example of the subgraph sampling. A guide-DAG G_1 (a) first is preprocessed to form G'_1 (b). After edge sampling, we construct the sampled-DAG S_1 (c) with the selected edges and vertices. Then, we perform operation sampling to choose the operation of each vertex to form our sampled architecture (d).

Algorithm 2 Sampling(G'_i)

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** $S_i(V_{S_i}, E_{S_i})$

(lines 8–13). The weight w_e of edge e is defined as

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

where α is the updating rate, and β is the baseline metrics. The weights are updated according to the difference between the evaluation metrics η and the baseline metrics β. We utilize an exponential function to boost the weight update. If the sampled model has a higher performance than the baseline, the edge weights will increase by this updating equation. Thus, the sample probabilities of these high-performance edges also increase in the following iteration. The baseline metrics β are set as the average evaluation metrics of all previously sampled models (line 14) to encourage the search process to explore higher-performing models across iterations. This process of sample and weight updating persists until the model performance reaches convergence.

TABLE I
BENCHMARK STATISTICS

Benchmarks	#nets			
	Min	Max	Median	Average
ISCAS'89 [25]	177	31650	650	3572
ITC'99 [26]	9635	206659	38879	57167
IWLS'05 [27]	581	101891	13134	28225
ISPD'15 [28]	29416	1293412	112877	253006

The probability sampling mechanism can encourage the exploration of different architectures in the search space since it covers all subgraphs and vertex operation combinations. Thus, our graph propagation enables a high flexibility of architectures in the search process. Also, graph propagation only takes about 0.5 days to identify high-quality architectures, indicating the weight updating in graph propagation can effectively guide the sampling process toward well-performance architectures. Therefore, our graph propagation method can efficiently find suitable architectures for routability prediction even with a huge search space.

V. Experimental Results

In this section, we first describe our experimental setups on dataset construction, feature construction and selection, and our NAS method along with training details. We then present our evaluations on two routability prediction benchmarks: 1) routing congestion and 2) DRC hotspot detection.

A. Experiment Setup

Dataset Construction: We construct a comprehensive dataset using 74 designs with largely varying sizes from multiple benchmarks. There are 29 designs from ISCAS'89 [25], 13 designs from ITC'99 [26], 19 other designs from Faraday and OpenCores in the IWLS'05 [27], and 13 designs from ISPD'15 [28] benchmark. The benchmark statistics are presented in Table I, which shows the minimum, maximum, median, and average number of nets in each benchmark design. The dataset covers designs with net numbers ranging from 177 to 1.29 million. Thus, the dataset can enable the model to learn feature representations with robust generalization to designs with various sizes. We apply Design Compiler for logic synthesis and Innovus [29] for physical

TABLE II
PERFORMANCE COMPARISON OF ROUTABILITY PREDICTION WITH OUR AUTOML-CRAFTED MODEL, ROUTENET [3], AND PROS [6]

Method	ROC-AUC on designs (#nets)				ROC-AUC on all 74 designs
	s344 (256)	s16850 (1.3k)	tv80 (13.5k)	mgc_fft_a (32.1k)	
RouteNet [3]	0.8781	0.7931	0.8135	0.7641	0.8035
PROS [6]	0.8864	0.8051	0.8384	0.8063	0.8246
AutoML-crafted	**0.9295**	**0.8699**	**0.8885**	**0.8791**	**0.8831**

design with the NanGate 45nm technology library [30]. For each design, multiple placement solutions are generated with different logic synthesis or physical design settings. Altogether 7000 placement solutions are generated from these 74 designs in industrial tools. Each layout is tiled by 1×1 μm^2 to generate features. Thus, each feature tensor uses the resolution according to its corresponding layout size. The raw features are collected at the post-placement stage, and the ground-truth DRC hotspots are available after detailed routing finishes.

The constructed dataset is randomly separated into two splits with different designs. The training set contains layouts from 51 designs, and the testing set are from 23 designs. Note that the designs are not overlap in two splits to demonstrate the transferability of our model. We use the results evaluated on the testing set as our comparison metrics.

Feature Extraction and NAS Training: For each layout in the target dataset, we follow Section IV-A to perform automated feature selection. After feature extraction, we adopt the NAS in Section IV-B to explore the search space defined for routability prediction. All experiments are performed on 4 NVIDIA TITAN RTX GPUs with Intel Xeon E5-2687W CPUs.

We employ the following hyperparameters to conduct model training for both feature selection and final evaluation in our experiments: we train our model for 128 epochs with Adam optimizer [31], a batch size of 32, and a fixed learning rate of 0.0005. To combat overfitting and improve generalization, we use a L2 weight decay of 10^{-5} and ReLU activation. For data augmentation, we employ random cropping with size 224×224 and random horizontal flipping. For NAS process, we set the number of epochs to 20 to enhance the search efficiency.

Baseline Methods: We adopt two representative routability prediction works [3], [6] as our baseline. We implement these hand-crafted routability features and model architectures by our own to compare with our AutoML framework.

B. DRC Hotspot Detection Results

To demonstrate the effectiveness of our AutoML framework, we compare our AutoML-crafted model with RouteNet [3] and PROS [6] trained by their own hand-crafted architectures and features. Note that our automated feature selection method extracts 21 features for our NAS method. In practice, chip designers typically focus on optimizing a specific design and care about the routability between different layout solutions of the same design. Therefore, the prediction performance within each design is important. In Table II, we first compare ROC-AUCs evaluated on the placement solutions of some specific designs with net numbers ranging from 256 to 32.1k. Our AutoML-crafted model clearly outperforms two baselines on these four designs. Then, we report the averaged ROC-AUC over all 74 designs to demonstrate the overall performance. Our AutoML-crafted model achieves better performance than RouteNet and PROS with 9.9% and 7.1% improvement, respectively. In addition, the ROC curves plotted in Fig. 10 show that under certain FPRs, our AutoML-crafted model consistently has higher TPRs compared to two baseline models.

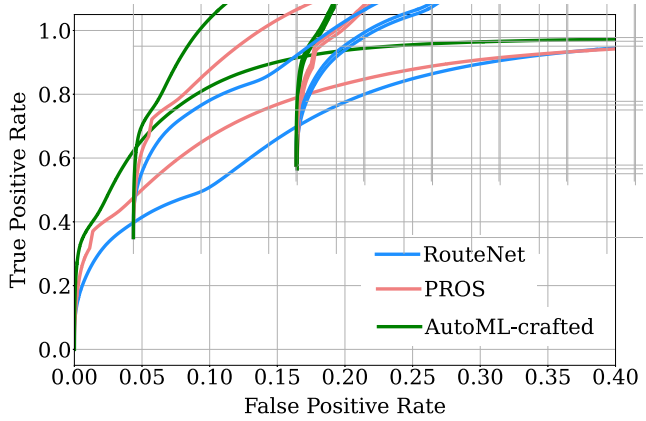


Fig. 10. ROC curves measured on all 74 designs with RouteNet [3], PROS [6], and our AutoML-crafted model.

We further benchmark our method on different feature numbers by using our automated feature selection to select varied top-k features based on the group norm with $k = \{4, 8, 21, 64\}$, where $k = 21$ is the automated-selected features, and $k = 64$ means training with the entire feature space. Based on this setting, we examine the performance contributions of our feature selection method and NAS method. First, to show the effectiveness of our feature selection method, we perform random selection as the baseline. We apply our selected features to our NAS method and other manually designed model architectures to make a comprehensive analysis. As shown in Table III, all models trained with our selected features can have better performance than the ones trained with random-selected features based on the same feature numbers. For example, when we use four features, the NAS-crafted model searched with our selected features can achieve 6.2% higher ROC-AUC thant the model searched with four random-selected features. In addition, all NAS-crafted models searched with our selected features (4, 8, 21) can outperform the model searched with the whole feature space (64 features). Thus, our feature selection can effectively remove redundant features and provide useful succinct feature sets with varied target feature numbers.

Next, to evaluate the performance improvement bringing from our NAS approach, we compare the NAS-crafted models with two human-crafted architectures by searching/training

TABLE III
PERFORMANCE COMPARISON OF ROUTABILITY PREDICTION BASED ON DIFFERENT MODEL ARCHITECTURES AND DIFFERENT FEATURE SELECTION METHODS WITH VARIED FEATURE NUMBERS TO DEMONSTRATE THE PERFORMANCE CONTRIBUTIONS OF OUR AUTOMATED FEATURE SELECTION AND THE NAS APPROACH

Model architecture	RouteNet [3]		PROS [6]		NAS-crafted	
#selected features	Random	Ours	Random	Ours	Random	Ours
4	0.8014	0.8511	0.8222	0.8466	0.8215	**0.8726**
8	0.8048	0.8541	0.8188	0.8527	0.8313	**0.8723**
21	0.8241	0.8567	0.8328	0.8425	0.8494	**0.8831**
64 (all)	0.8496		0.8585		0.8695	

TABLE IV
SEARCH TIME COMPARISON OF OUR NAS METHOD WITH DIFFERENT FEATURE NUMBERS {4, 8, 21, 64}

#features	4	8	21	64
Search time of NAS (hr)	8.66	11.84	20.01	113.18

with our selected features. According to Table III, when trained with our 21 features, the NAS-crafted model demonstrates a high ROC-AUC of 0.8831, while RouteNet can only have ROC-AUC of 0.8567. Overall, our NAS-crafted models can outperform two human-crafted architectures under all different feature sets, indicating that our NAS approach can effectively boost the model performance.

In addition, we present the search time of our NAS under different feature numbers (4/8/21/64) given the same model exploration budget. As Table IV shows, an increase in the feature numbers leads to longer search time. We posit this is due to the expansion of model parameters and the severe data input/output (I/O) overhead. This result indicates that our feature selection can accelerate the NAS process by 6.1 to 12.1× compared to searching with all 64 features. Also, our feature selection only takes about 5 h. Thus, the entire AutoML process can be completed in about 0.6 to 1.0 days to automatically build well-performing models for routability prediction. In addition, this presents a significant improvement in model development efficiency over manual model design, which can take weeks or even months.

Finally, we investigate the performance when using a different number of sampled-DAGs in our search space. In detail, we set the sampled-DAG number to three in our search space, which contains one sampled-DAG between every two downsampling layers and do not have parallel DAG structures. Based on this search space, the NAS-crafted model achieve a slightly lower ROC-AUC of 0.8703 compared to the model searched with our original search space composed of six DAGs, which has ROC-AUC of 0.8831. The observation suggests the importance of maintaining the original DAG number and the parallel structure in the search space to ensure the model performance.

VI. DISCUSSION

In this section, we first present an analysis of our selected features. Then, we discuss the architectures of our NAS-crafted models searched with different numbers of features. To provide a more comprehensive analysis, we also compare our NAS-crafted models with manually crafted models (e.g., RouteNet [3], PROS [6], and cGAN [4]). By examining both feature selection and architecture design generated by our AutoML framework, we aim to offer valuable insights for the development of future routability estimators.

A. Automatic Selected Feature Analysis

We list the 21 features selected by our automated feature selection method according to the rank.

1) Bounding boxes.
2) Cell density.
3) Pin density (clock tree).
4) RUDY.
5) Cell density (clock tree).
6) Pairwise flight lines.
7) Vertical net density.
8) Pin density.
9) Pin accessibility (DFF).
10) Pin accessibility.
11) Bounding boxes (nets with size $<$ 4k).
12) Source-sink flight lines.
13) RSMT RUDY.
14) Macro density.
15) Cell density (DFF).
16) Horizontal net density (nets with size $<$ 1k).
17) RUDY (nets with size $<$ 0.1k).
18) Pin density (nets with size $<$ 4k).
19) MST flight lines.
20) Pin density (nets with size $<$ 1k).
21) RUDY (nets with size $<$ 1k).

Note that the feature name is followed by the cell/net properties, and features without specified properties are built by all cells/nets. For example, pin density (clock tree) represents the density of cell pins belonging to clock tree buffers.

The most important feature is the bounding boxes of all nets. Because the router typically employs L-shape routing method on most nets to enhance routing efficiency, bounding boxes provide a good estimation of the routing solution. In addition, this feature has never been adopted in previous works. This suggests that our feature extraction can discover new features that go beyond the traditional sense of routability features. Then, our method selects the cell density and the pin density (clock tree). Cell density is applied in all routability prediction works. It is interesting that the pin density of clock tree buffers (rank 3) is more important than the pin density of all cells (rank 8). We posit that clock tree nets usually are the most complicated nets since all functional blocks require clock signals to operate. Thus, pins of clock tree buffers could have a higher chance of causing DRC hotspot than normal pins. Next, our method sets RUDY as the forth important feature, which is

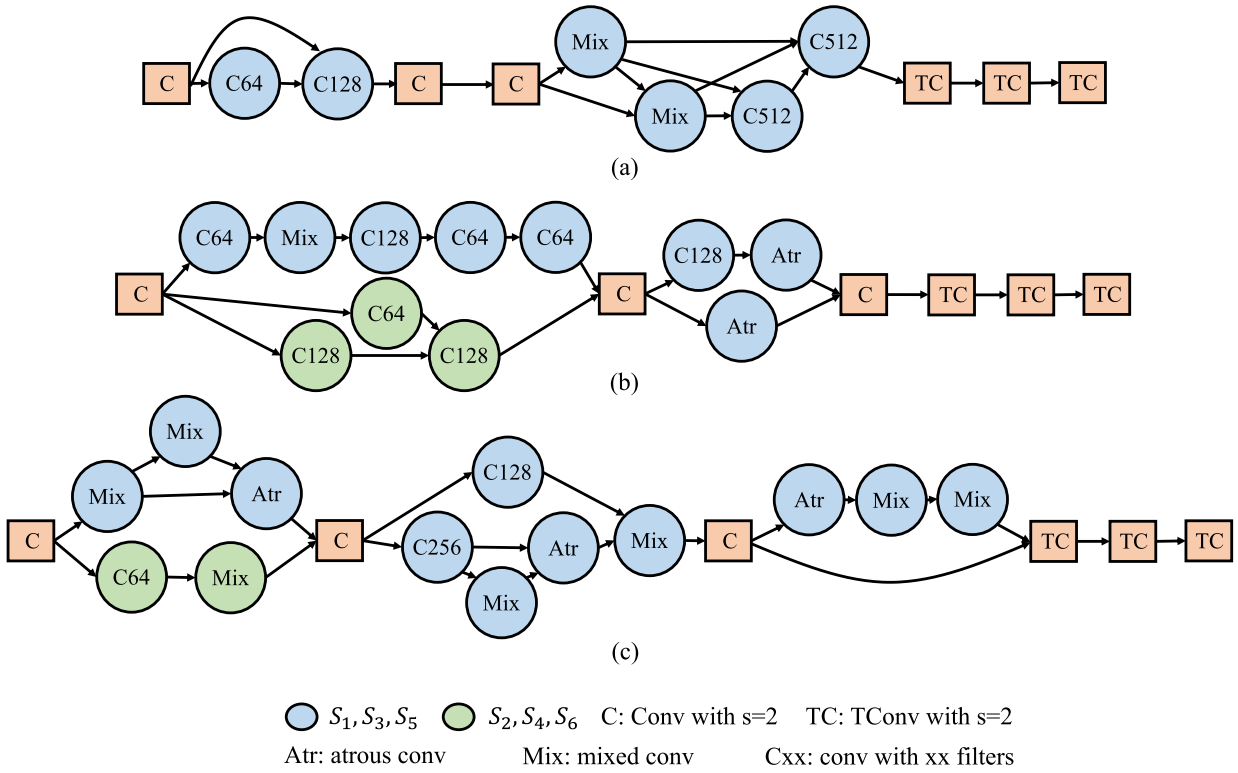


Fig. 11. NAS-crafted models searched with (a) 4, (b) 8, and (c) 21 features for routability prediction.

also widely applied in previous works. According to Tables III and IV, the NAS-crafted model searched with top-4 features only exhibit little performance degradation but with shorter search time compared to the model searched with 21 features. Thus, researchers who prioritize development efficiency can focus on these 4 features to save time without compromising much on the model performance. In addition, these top-4 features are repeatedly chosen with different net/cell properties in the top-21 features, which highlights the importance of them to facilitate routability prediction.

Among the remaining features, our method selects the other 6 types of net density-based features, including pairwise and source-sink flight line, vertical/horizontal net density, the RSMT RUDY, and the MST flight line. Also, RUDY and pin density features with different net sizes are included, indicating the importance of these type of features. In addition, almost all cell density-based features are included. By using these 21 features, our NAS-crafted model can have the best performance based on the results in Table III. As a result, we believe that our feature selection results can serve as a valuable benchmark for future routability estimator development.

B. NAS-Crafted Model Analysis

We compare and analyze the NAS-crafted models searched using different number feature numbers, including 4, 8, and 21, as illustrated in Fig. 11. This analysis aims to provide valuable insights about the future development of routability estimators that target different features.

In sampled-DAGs 1 and 2, we observe that the model with 4 features only adopts two standard convolutional layers, while the models with 8 and 21 features utilize wider and more convolutional layers. Specifically in the model with 8 features, sampled-DAGs 1 and 2 are the stage with the most operations. This suggests that this stage is crucial to identify DRC hotspot information in this resolution where the input tensor is reduced by $2\times$. In sampled-DAG 3 and 4, we find an interesting trend: the model with four features does not include any operation, while the models with 8 and 16 features use several atrous and mixed convolution layers. This observation indicates that when the input resolution is reduced by 4x, the model with 4 features does not require any operation to process feature information. However, the models with more features require large-kernel convolution layers to extract large-scale hotspot patterns. In sampled-DAG 5 and 6, the models with 4 and 21 features employ structures with lots of mix convolutions and rich edge connections. These structures can enhance feature interactions among operations to help capturing the complex hotspot location information. The model with 8 features does not include any operation in this stage, meaning the prior stages have provided enough expressive power for identifying DRC hotspots. Also, keeping model compact could prevent overfitting during prediction.

After detailing each sampled-DAG, we provide an overview analysis of three NAS-crafted models. For the model with 4 features, our NAS method only assigns 6 operations to predict routability. In contrast, the models searched with 8 and 21 features require 11 and 13 operations, respectively. The increased operation numbers for models with more features are due to the need for additional operations to effectively express the feature representation toward DRC hotspot. In summary, three NAS-crafted models exhibit large architecture

differences. This result highlights the need of designing unique architectures for different feature sets and the advantage of using our NAS method. In addition, we observe that the NAS highly prefers a complex combination of mixed convolution layers, which effectively learns from both small-scale and large-scale input patterns. Such structure reflects the nature of DRC hotspot detection that both local and global patterns of a layout influences the routability at each point.

Human developers can hardly explore structures that are similar to the NAS-crafted models. Most human-crafted models only support a limited number of operators (typically regular convolution), and thus have limited ability to learn the large-scale input patterns. They also adopt highly hierarchical architectures that lack the ability to aggregate different levels of features. In contrast, our NAS method supports operators that process features very differently. The variation of vertex operations on the branches greatly increase the diversity of feature representations that can be explored by our NAS method. Moreover, our NAS method can construct a number of scalable parallel branches and explore flexible interactions among them, which is inherent in the topology of the guide-DAGs. Therefore, our NAS method can extract feature representations of large and small-scale patterns at the same time and find the best interactions of features, which is critical to improving the accuracy of routability prediction.

C. Future Direction

In this study, we introduce an AutoML framework that target on optimizing the model accuracy for routability prediction. Minimizing the model inference time could be another important objective when applying the model in real optimization scenarios. Because PROS [6] is used in optimizing global routing, we compare the floating point operations per second (FLOPS) and the number of model parameters of PROS and our AutoML-crafted model. Our model has 14905M FLOPS and 3M parameters. In comparison, PROS has 170052M FLOPS and 206M parameters. Thus, our model not only has better performance but also exhibits better-inference efficiency than PROS, making it more suitable for optimization. In the future, we can incorporate the number of parameters or FLOPS as a penalty in the NAS objective to optimize the model inference time and the accuracy at the same time.

VII. Conclusion

In this work, we develop an AutoML framework to automate the ML model development for routability prediction. We design an automated ML-based feature selection method that efficiently identify suitable raw features for model inputs. We propose an NAS method that supports a large search space with various operations and highly flexible connections and automates the design of ML architectures. To the best of our knowledge, this is the first research effort that automates the feature selection for EDA. Our model, which only requires 1 day to be automatically generated, proves to outperform previous human-crafted routability estimators with the maximum improvement of 9.9%. In addition, we provide an in-depth analysis of our automatically selected features and model architectures to benefit the future development of routability estimators. Although we focus on routability prediction, our framework could be general. Our automated feature selection method can be applied to the model development for any ML applications to choose proper raw features. Also, given the similarities in solutions between routability prediction and other essential EDA problems, e.g., IR drop estimation, clock tree prediction, lithography hotspot detection, optical proximity correction, etc., our NAS method could ultimately benefit the solving of these problems. The AutoML framework proposed in this work paves the way for a new research direction, toward the automation of ML model development for EDA applications.

References

[1] W.-T. J. Chan, P.-H. Ho, A. B. Kahng, and P. Saxena, "Routability optimization for industrial designs at sub-14nm process nodes using machine learning," in *Proc. ACM Int. Symp. Phys. Design*, 2017, pp. 15–21.

[2] A. F. Tabrizi et al., "A machine learning framework to identify detailed routing short violations from a placed netlist," in *Proc. ACM/IEEE Design Autom. Conf.*, 2018, pp. 1–6.

[3] Z. Xie et al., "RouteNet: Routability prediction for mixed-size designs using convolutional neural network," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2018, pp. 1–8.

[4] C. Yu and Z. Zhang, "Painting on placement: Forecasting routing congestion using conditional generative adversarial nets," in *Proc. ACM/IEEE Design Autom. Conf.*, 2019, p. 219.

[5] R. Liang et al., "DRC hotspot prediction at sub-10nm process nodes using customized convolutional network," in *Proc. ACM Int. Symp. Phys. Design*, 2020, pp. 135–142.

[6] J. Chen, J. Kuang, G. Zhao, D. J.-H. Huang, and E. F. Young, "PROS: A plug-in for routability optimization applied in the state-of-the-art commercial EDA tool using deep learning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–8.

[7] T.-C. Yu et al., "Pin accessibility prediction and optimization with deep learning-based pin pattern recognition," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–6.

[8] M. B. Alawieh, W. Li, Y. Lin, L. Singhal, M. A. Iyer, and D. Z. Pan, "High-definition routing congestion prediction for large-scale FPGAs," in *Proc. IEEE 25th Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2020, pp. 26–31.

[9] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowl. Based Syst.*, vol. 212, Jan. 2021, Art. no. 106622.

[10] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8697–8710.

[11] Z. Lu et al., "NSGA-Net: Neural architecture search using multi-objective genetic algorithm," in *Proc. Genet. Evol. Comput. Conf.*, 2019, pp. 419–427.

[12] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," 2018, *arXiv:1802.03268*.

[13] C. Li, M. Xie, C.-K. Koh, J. Cong, and P. H. Madden, "Routability-driven placement and white space allocation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 5, pp. 858–871, Nov. 2007.

[14] Z. Qi, Y. Cai, and Q. Zhou, "Accurate prediction of detailed routing congestion using supervised data learning," in *Proc. IEEE 32nd Int. Conf. Comput. Design (ICCD)*, 2014, pp. 97–103.

[15] W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath, and K. Samadi, "BEOL stack-aware routability prediction from placement using data mining techniques," in *Proc. IEEE 34th Int. Conf. Comput. Design (ICCD)*, 2016, pp. 41–48.

[16] B. Wang et al., "LHNN: Lattice hypergraph neural network for VLSI congestion prediction," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, 2022, pp. 1297–1302.

[17] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. 18th Int. Conf. Med. Image Comput. Comput.-Assist. Intervent. (MICCAI)*, Munich, Germany, Oct. 2015, pp. 234–241.

[18] C. Liu et al., "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 82–92.
[19] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv:1812.00332*.
[20] H.-P. Cheng et al., "SwiftNet: Using graph propagation as meta-knowledge to search highly representative neural architectures," 2019, *arXiv:1906.08305*.
[21] C.-C. Chang et al., "Automatic routability predictor development using neural architecture search," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2021, pp. 1–9.
[22] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *J. Roy. Stat. Soc. B Stat. Methodol.*, vol. 68, no. 1, pp. 49–67, 2006.
[23] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017, *arXiv:1706.05587*.
[24] M. Tan and Q. V. Le, "MixConv: Mixed depthwise convolutional kernels," 2019, *arXiv:1907.09595*.
[25] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 1989, pp. 1929–1934.
[26] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Des. Test Comput.*, vol. 17, no. 3, pp. 44–53, Jul.–Sep. 2000.
[27] C. Albrecht. "IWLS 2005 benchmarks." 2005. [Online]. Available: http://www.iwls.org
[28] I. S. Bustany, D. Chinnery, J. R. Shinnerl, and V. Yutsis, "ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement," in *Proc. Symp. Int. Symp. Phys. Design*, 2015, pp. 157–164.
[29] Cadence. "Innovus implementation system." 2021. [Online]. Available: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html
[30] "NanGate 45nm open cell library." Accessed: Sep. 1, 2021. [Online]. Available: https://si2.org/open-cell-library/
[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

Chen-Chia Chang received the B.S. degree in electrical engineering from National Taiwan University, Taipei City, Taiwan, in 2020. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the Computational Evolutionary Intelligence Lab, Duke University, Durham, NC, USA, advised by Prof. Yiran Chen.

His research interests are focused on electronic design automation and machine learning algorithms.

Dr. Chang has received the ASP-DAC 2023 Best Paper Award.

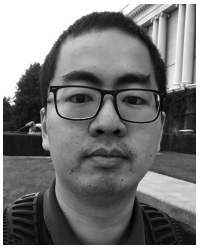
Jingyu Pan received the B.Eng. degree from Zhejiang University, Hangzhou, China, in 2020. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, Duke University, Durham, NC, USA.

His research interests include machine learning applications in electronics design automation, and VLSI circuits and systems.

Zhiyao Xie (Member, IEEE) received the B.Eng. degree from The City University of Hong Kong, Hong Kong, in 2017, and the Ph.D. degree from Duke University, Durham, NC, USA, in 2022.

He is an Assistant Professor with the Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Hong Kong. His research interests include machine learning algorithms for EDA and VLSI design.

Dr. Xie has received multiple prestigious awards, including the IEEE/ACM MICRO 2021 Best Paper Award, the ACM SIGDA SRF Best Research Poster Award 2022, the ASP-DAC 2023 Best Paper Award, the ACM Outstanding Dissertation Award in EDA 2023, the EDAA Outstanding Dissertation Award 2023, and the 2023 Early Career Award from Hong Kong Research Grants Council.



Tunhou Zhang (Graduate Student Member, IEEE) received the B.Eng. degree from Fudan University, Shanghai, China, in 2018, and the M.S. degree from Duke University, Durham, NC, USA, in 2020, where he is currently pursuing the Ph.D. degree in electrical and computer engineering.

His research interests lie in automated machine learning, especially in improving neural architecture search techniques and applying NAS toward real-world applications, such as recommender systems, computer vision, and electronic design automation.

Jiang Hu received the B.S. degree in optical engineering from Zhejiang University, Hangzhou, China, in 1990, the M.S. degree in physics from the University of Minnesota at Duluth, Duluth, MN, USA, in 1995, and the Ph.D. degree in electrical engineering from the University of Minnesota, Minneapolis, MN, USA, in 2001.

He is a Professor with the Department of Electrical and Computer Engineering, Texas A&M University at College Station, College Station, TX, USA. His research interests include EDA, computer architecture, and hardware security.

Dr. Hu received best paper awards at DAC 2001, ICCAD 2011, MICRO 2021, and ASPDAC 2023. He served as the General Chair of the ACM International Symposium on Physical Design 2012 and the Technical Program Co-Chair of the ACM/IEEE Workshop on Machine Learning CAD 2023.

Yiran Chen (Fellow, IEEE) received the B.S. and M.S. degrees from Tsinghua University, Beijing, China, in 1998 and 2001, respectively, and the Ph.D. degree from Purdue University, West Lafayette, IN, USA, in 2005.

After five years in the industry, he was an Assistant Professor with the University of Pittsburgh, Pittsburgh, PA, USA, in 2010, and was promoted to an Associate Professor with tenure in 2014, holding a Bicentennial Alumni Faculty Fellow. He is currently the John Cocke Distinguished Professor of Electrical and Computer Engineering with Duke University, Durham, NC, USA, and serving as the Director of the NSF AI Institute for Edge Computing Leveraging the Next-Generation Networks (Athena). His group focuses on the research of new memory and storage systems, machine learning and neuromorphic computing, and mobile computing systems.