Automated and Agile Design of Layout Hotspot Detector via Neural Architecture Search

Zihao Chen[1], Fan Yang[1*], Li Shang[2], Xuan Zeng[1*]
[1]State Key Lab of ASIC & System, School of Microelectronics, Fudan University, Shanghai, China
[2]China and Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University, Shanghai, China

***Abstract*—This paper presents a neural architecture search scheme for chip layout hotspot detection. In this work, hotspot detectors, in the form of neural networks, are modeled as weighted directed acyclic graphs. A variational autoencoder maps the discrete graph topological space into a continuous embedding space. Bayesian Optimization performs neural architecture search in this embedding space, where an architecture performance predictor is employed to accelerate the search process. Experimental studies on ICCAD 2012 and ICCAD 2019 Contest benchmarks demonstrate that, the proposed scheme significantly improves the agility of previous neural architecture search schemes, and generates hotspot detectors with competitive detection accuracy, false alarm rate, and inference time.**

***Index Terms*—layout hotspot detection, neural architecture search, Bayesian Optimization, variational autoencoder**

I. Introduction

With the increasing complexity of lithography systems and process variation, manufacturing defects caused by sensitive layout patterns, so-called layout hotspots, have become inevitable. However, classical lithography simulation is highly time-consuming [1]. Therefore, layout hotspot detection with high accuracy and efficiency, is essential but challenging.

Two hotspot detection techniques have been extensively studied recently: pattern matching [2], [3] and machine learning (ML) approaches [4], [5]. Pattern matching approaches detect hotspots via explicit pattern matching given known defect patterns, fast but inapplicable to unseen patterns. ML approaches, however, can detect unseen hotspots with adequately trained classifiers. Recent works on ML approaches, especially neural network (NN) approaches such as convolutional neural networks (CNNs) [5] and binarized neural networks [6], have been very promising due to considerable generalizability [1].

However, neural architecture design to enable predominant hotspot detection requires extensive design experience and domain expertise [7]. Meanwhile, model parameter tuning is also time-consuming. These drawbacks hinder manually designed NN-based hotspot detectors from moving toward large-scale practical use. Therefore, an automated NN design approach, so-called neural architecture search (NAS), has become an alternative for hotspot detection tasks.

A NAS scheme automatically designs NNs by searching neural architecture topology. Recent works on NAS [8], [9] demonstrate their capability to generate NNs with high performance. NAS was introduced into hotspot detection tasks for the first time in [10], whose long search time, however, limits the practical use of conventional NAS solutions.

The high time cost of NAS comes from two aspects, i) the vast architectures search space and ii) the high training cost when evaluating candidate architectures. Variational autoencoder (VAE) techniques [11] can be adopted to map the search space from discrete to continuous and back. Circuitous evaluation strategies using an architecture performance predictor [12] could accelerate the search process.

This work presents a NAS scheme for the automated design of NN-based hotspot detectors. Bayesian Optimization (BO) is applied as the top-level algorithm, for fully exploring the search space while exploiting potential architectures. Neural architectures are modeled as weighted directed acyclic graphs (DAGs) in a constrained search space. A VAE for weighted DAGs is used to support the execution of BO, namely the search process, during which a performance predictor composed of multi-layer perceptron (MLP) is iterated to accelerate the search process for optimal architectures. Search results are finally decoded into neural architectures to execute hotspot detection tasks. Experiments conducted on ICCAD 2012 [13] and ICCAD 2019 [7] Contest benchmarks demonstrate that, the proposed scheme can generate high-performance hotspot detectors and significantly improve the design efficiency.

The rest of this paper is organized as follows. Preliminaries on the hotspot detection and relevant algorithmic basis are introduced in Section II. The NAS-based detection method is proposed in Section III. Section IV focuses on implementation details. Experimental settings, results, and analysis are presented in Section V. Section VI concludes this paper.

II. Preliminaries

A. Problem Formulation

Open or short circuit failures come from different variations in the lithographic process. Hotspots refer to layout patterns sensitive to these process variations. Hotspot detection is a binary classification task based on layout clip datasets with only hotspot and non-hotspot labels. The objective is to classify clips accurately, or more specifically, to identify real hotspots while avoiding mistaking non-hotspots for hotspots [1].

In hotspot detection tasks, two metrics are adopted, i.e., detection accuracy and false alarm rate, quantified by the confusion matrix in Table I. As shown in (1) and (2), detection accuracy, or so-called *recall*, is the rate of hotspots identified as hotspots. In contrast, the false alarm rate, abbreviated as *FA*, is the rate of non-hotspots identified as hotspots.

$$recall = \frac{\#TP}{\#TP + \#FN}, \tag{1}$$

$$FA = \frac{\#FP}{\#TN + \#FP}. \tag{2}$$

*Corresponding authors: {yangfan, xzeng}@fudan.edu.cn.

978-3-9819263-7-8/DATE23/© 2023 EDAA

The objective is to maximize *recall* while minimizing *FA*, implemented by maximizing the mixed metric F_1 in (3).

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}, \tag{3}$$

where *precision* is defined in (4):

$$precision = \frac{\#TP}{\#TP + \#NP}. \tag{4}$$

TABLE I
CONFUSION MATRIX OF HOTSPOT DETECTION PROBLEM

Prediction	**Ground Truth**	
	Non-hotspot	Hotspot
Non-hotspot	$\#TN$	$\#FN$
Hotspot	$\#FP$	$\#TP$

B. Neural Architecture Search

A NAS scheme aims to find the optimal neural architecture based on a specific dataset and task. It comprises search space, performance evaluation strategy, and search algorithm.

1) Search space: Neural architectures modeled as DAGs lead to the discrete topological search space [9], [14], [15]. Some innovative works [8], [9] try to map the search space into continuous but fall into the dilemma that embeddings cannot be translated back into the topological space.

2) Performance evaluation strategy: A conventional performance evaluator outputs the validation accuracy of an architecture trained from scratch to convergence [15], which is exceptionally time-consuming. Subsequently, proxy NN models with fewer layers and proxy tasks with minor scales are proven economic [8], [9], [14]. However, these strategies still suffer from the stubborn problem of repeated training.

3) Search algorithm: Mainstream search algorithms include Evolutionary Algorithm [14], Gradient Descent (GD) [8], Reinforcement Learning [15], and Bayesian Optimization [9]. This paper uses BO as the top-level search algorithm.

As shown in Fig. 1, in a typical NAS workflow, given initial architectures in the search space randomly, their metrics are evaluated and fed back to the algorithm to guide the search for candidates. The optimal architecture recorded is finally saved.

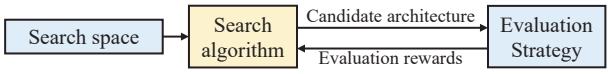


Fig. 1. A typical NAS workflow.

C. Bayesian Optimization

Bayesian Optimization is a classical black-box optimization technique. Without loss of generality, the optimization problem is denoted in (5) as maximization for $y = f(\mathbf{x})$.

$$(\mathbf{x}^*, y^*) = \max_{\mathbf{x}} f(\mathbf{x}). \tag{5}$$

The BO process is summarized in Algorithm 1. In an iteration, the Gaussian Process (GP) model is employed to estimate $f(\mathbf{x})$ from the existing data pool. Based on the prior distribution originating from this GP model, an acquisition function $Acq(\mathbf{x})$ is constructed and optimized to generate a candidate pair, which is finally merged into the data pool.

Algorithm 1 Bayesian Optimization

1: Sample initial data pool $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ randomly
2: **for** $t \leq iterations$ **do**
3: Construct the GP model through $\mathcal{D}$
4: Generate candidate $\mathbf{x}_t$ by optimizing $Acq(\mathbf{x})$
5: Get $y_t = f(\mathbf{x}_t)$ and update data pool $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}_t, y_t)\}$
6: **end for**
7: **return** the optimal pair $(\mathbf{x}^*, y^*)$ recorded

D. Variational Autoencoder for Graphs

The BO algorithm relies on a well-characterized continuous space. As shown in Fig. 2, variational autoencoders can construct a mutual map between graph topological space and continuous embedding space [11]. Without loss of generality, for a graph $g = (V, E)$, we mark the adjacency matrix as $\mathbf{A}$, the node feature matrix as $\mathbf{F}$, and the number of nodes as N.

1) Encoder: The embedding $\mathbf{z}$ for g, relys on Gaussian vector $\mathbf{z}^i$ for each node $v \in V$. The mean and variance of $\mathbf{z}$ are extracted by a pair of graph convolutional networks (GCNs):

$$\boldsymbol{\mu} = \mathrm{GCN}_{\mu}(\mathbf{F}, \mathbf{A}), \tag{6}$$

$$\log \boldsymbol{\sigma} = \mathrm{GCN}_{\sigma}(\mathbf{F}, \mathbf{A}). \tag{7}$$

Subsequently, the joint distribution of $\mathbf{z}$ is produced by (8). For brevity, the symbol g here includes metrics $\mathbf{A}$ and $\mathbf{F}$ above.

$$q(\mathbf{z}|g) = \prod_{i=1}^{N} \mathcal{N}\left(\mathbf{z}^i \middle| \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2 \cdot \mathbf{I}\right). \tag{8}$$

2) Decoder: The decoder is usually an inner product layer:

$$p(g|\mathbf{z}) = \prod_{i=1}^{N} \prod_{j=1}^{N} p\left(A_{ij} \middle| \mathbf{z}^i\right), \tag{9}$$

where

$$p\left(A_{ij} = 1 \middle| \mathbf{z}^i, \mathbf{z}^j\right) = \mathrm{sigmoid}[(\mathbf{z}^i)^\top \mathbf{z}^j]. \tag{10}$$

3) Loss function: The following hybrid loss function is minimized in the VAE training phase:

$$L_{VAE} = \mathbb{E}_{q(\mathbf{z}|g)}\left[-\log p(g|\mathbf{z})\right] + \beta \cdot KL[q(\mathbf{z}|g) \,||\, p(\mathbf{z})], \tag{11}$$

where $KL[q(\cdot) \,||\, p(\cdot)]$ is the Kullback-Leibler divergence between the posterior approximation $q(\cdot)$ and the prior $p(\cdot)$. The first term is related to the similarity between reconstructed and original graphs. The second term regularizes the latent space for sampling $\mathbf{z}$ from a prior $p(\mathbf{z})$ instead of $q(\mathbf{z}|g)$. β is a tunable weight hyperparameter.

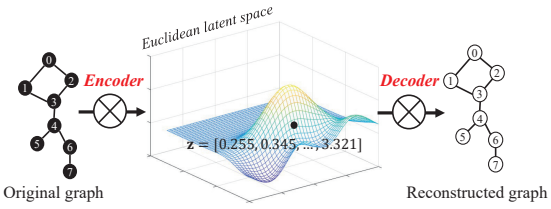


Fig. 2. A typical encoding and decoding workflow in VAE.

III. Proposed Approaches

This section presents the proposed NAS scheme for hotspot detection bottom-up. First, Section III-A defines the topological space for neural architectures. Section III-B then adopts a VAE for weighted DAGs to map this space into continuous. In this embedding space, with an advanced NN performance evaluation strategy in Section III-C, the BO-based NAS scheme is proposed in Section III-D for the automated design of hotspot detectors. Section III-E summarizes the overall workflow.

A. Topological Space of Neural Architectures

Considering the essence (the binary classification of images) of hotspot detection, This paper explicitly uses CNN as a hotspot detector by weighted DAGs. The following hypotheses define the topological space of NNs, referring to [8] and [11].

- A CNN is stacked from primary modules, so-called *cells*. Each cell maps a tensor from $c \times h \times w$ to $c' \times h' \times w'$. Cells of the same type share their parameters.
- There are two types of cells, *normal cell* and *reduction cell*. In normal cells, convolution operator's stride is 1, i.e., $h = h'$ and $w = w'$. In reduction cells, convolution operator's stride is 2, i.e., $h = 2h'$ and $w = 2w'$.
- Each cell is a weighted DAG, where each node represents a calculated state while each edge represents an operator noted by its weight. Table II shows that 7 optional operators correspond to different weights.
- The internal structure for a single cell is shown in Fig. 3(a). There are three types of nodes: i) input nodes *input1* and *input2*, succeeding output tensors from two closest previous cells; ii) intermediate nodes from 0 to 3; iii) output node *output*. Nodes with multiple inputs are computed as the element-wise sum of input edges. Each node can connect to its successors except the output-input straight connection. All intermediate nodes are concatenated as *output* by 4 fixed edges (solid lines) while as many as 14 optional edges are marked as dashed lines, a few of which are fixed in practice.
- When stacking for the overall neural network, as shown in Fig. 3(b), reduction cells only appear in the trisection positions while normal cells occupy the rest positions.

The search space, noted as $\mathscr{G}$, is thus constrained. Denote single cell's topological space as $\mathscr{G}_{cell}$, $\mathscr{G}_{cell} \subset \mathscr{G}$. A hotspot detector $g \in \mathscr{G}$ is expressed by two cells, as shown in (12).

$$g = \mathcal{S}(g_n, g_r), \tag{12}$$

where $\mathcal{S}$ represents the stacking operator; subscripts n and r represent shorthand for cell type *normal* and *reduction*, respectively.

TABLE II
Operators and Weights in the Search Space

Operator Option	Weight
3×3 separable convolution	1
5×5 separable convolution	2
3×3 dilated separable convolution	3
5×5 dilated separable convolution	4
3×3 max pooling	5
3×3 average pooling	6
identity	7

B. VAE for Weighted DAGs

To enable Bayesian Optimization, a mutual map $\mathcal{M}$ between the topological space $\mathscr{G}$ and an m-dimensional Euclidean space $\mathscr{R}^m$, should be constructed.

$$\mathcal{M} : \mathscr{G} \leftrightarrow \mathscr{R}^m. \tag{13}$$

Considering the GP model in BO, VAE based on Gaussian distribution is a fine choice. A VAE is modified from [16] to handle weighted DAGs.

1) Encoder: Different from (6) and (7), the hidden state of each node is calculated in turn by aggregating the hidden states of all predecessors along the information transfer direction in the DAG. Specifically, the states of node v's all predecessors are aggregated to the input state $\mathbf{h}_v^{in}$ by (14), and updated to the output state $\mathbf{h}_v^{out}$ by a gated recurrent unit (GRU) in (15).

$$\mathbf{h}_v^{in} = \sum_{V_p} \mathrm{g}\left(e_{V_p \to v} \times \mathbf{h}_{V_p}\right) \odot \mathrm{m}\left(e_{V_p \to v} \times \mathbf{h}_{V_p}\right), \tag{14}$$

$$\mathbf{h}_v^{out} = \mathrm{GRU}_{enc}\left(w_v,\ \mathbf{h}_v^{in}\right), \tag{15}$$

where $\mathrm{g}(\cdot)$ is the gating network while $\mathrm{m}(\cdot)$ is the mapping network; e represents the edge weight between node v and its predecessors V_p; and w_v represents the weight in Table II.

Finally, the DAG's hidden state $\mathbf{h}_{enc}$ is produced after all nodes have been traversed. Equations (6) and (7) are then adapted in (16) and (17).

$$\boldsymbol{\mu} = \mathrm{MLP}_{\mu}(\mathbf{h}_{enc}), \tag{16}$$

$$\log \boldsymbol{\sigma} = \mathrm{MLP}_{\sigma}(\mathbf{h}_{enc}). \tag{17}$$

To avoid the numerical overflow of $\boldsymbol{\sigma}$ in (17), a *sigmoid*-like constraint function is adopted here. The latent vector $\mathbf{z}$ is sampled from $q(\mathbf{z}|g)$ similar to (8).

2) Decoder: A gated recurrent unit GRU_{dec} followed by MLP_{dec} updates the embedding and calculate the prior p_{edge}:

$$\mathbf{h}_{dec} = \mathrm{GRU}_{dec}(\mathbf{z}), \tag{18}$$

$$p_{edge}\left(g|\mathbf{z}\right) = \mathrm{MLP}_{dec}(\mathbf{h}_{dec}). \tag{19}$$

$g_{rec} = (V, E_{rec})$ is reconstructed by sampling from $p_{edge}(\cdot)$.

In the training process, (8) and (19) are substituted into (11). L_{VAE} is minimized to update the network till convergence. See Section IV-A for details on the VAE's training.

In practice, the bidirectional map $\mathcal{M}$ is implemented on a single cell's search space $\mathscr{G}_{cell}$. The encoding and decoding process for a hotspot detector g, marked by $\mathcal{M}_{enc}(\cdot)$ and $\mathcal{M}_{dec}(\cdot)$, are generalized by (20) and (21), respectively.

$$\mathbf{z} = \mathcal{M}_{enc}(g) = [\mathcal{M}_{enc}(g_n), \mathcal{M}_{enc}(g_r)] = [\mathbf{z}_n, \mathbf{z}_r], \tag{20}$$

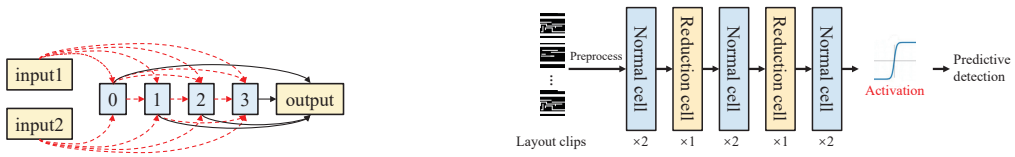
$$g_{rec} = \mathcal{M}_{dec}(\mathbf{z}) = \mathcal{S}(\mathcal{M}_{dec}(\mathbf{z}_n), \mathcal{M}_{dec}(\mathbf{z}_r)), \tag{21}$$

where $\mathbf{z} \in \mathscr{R}^m$ and $\mathbf{z}_n, \mathbf{z}_r \in \mathscr{R}^{m/2}$; $g \in \mathscr{G}$ and $g_n, g_r \in \mathscr{G}_{cell}$.

C. Performance Evaluation of Detectors

The topological space of the hotspot detectors has been defined and mapped into continuous. The performance evaluation strategy should be draughted afterward. The metric F_1 is utilized to evaluate the searched detectors.

A fundamental performance evaluation is conducted in the final testing case. The optimal neural architecture is trained for epochs until the best performance is achieved, that is, the final hotspot detector.



(a) (b)

Fig. 3. The topological architecture of hotspot detectors. Fig. 3(a) presents the single proposed cell's inside structure, where nodes are state tensors and edges (in red) are operators. Edges connected to *output* is fixed; certain edge types (operators) are not marked here for generality. Fig. 3(b) presents the macro architecture of a hotspot detector. There is a total of 8 cells here; only the 3rd and 6th ones are reduction cells. Layout clips are preprocessed and fed into this 8-layer CNN; after a *sigmoid*-like activation function, the CNN predicts whether an input clip is a hotspot or non-hotspot.

However, during the search process requiring agility, the following strategies are used to estimate candidates' final F_1.

- Candidates are only trained for one epoch before verifying F_1 as the ground-truth score on the validation set, instead of training from scratch to convergence like [9], [14], [15].
- The search process is still relatively slow due to repeated training. It is noticed that [12], under the same task, dataset, and training method, a correlation between the neural architecture and its performance is latent but undeniable. Meanwhile, neural architectures have been mapped to latent vectors according to (20). Therefore, a further estimation of F_1 is adopted by conducting an MLP-based performance predictor. As shown in (22), the embedding $\mathbf{z}$ is mapped to a scalar predictive score $\hat{s}$.

$$\hat{s} = \mathcal{P}(\mathbf{z}) = \mathrm{MLP}_{pred}(\mathbf{z}), \tag{22}$$

where a *sigmoid* activation function matches the range of F_1. The embedding $\mathbf{z}$ and the corresponding ground-truth score F_1, are employed to train the predictor. Moreover, it is more practical to focus on the relative merits of architectures rather than the absolute score values. A ranking error function could be adopted instead of the standard Mean Squared Error (MSE) or Mean Absolute Error (MAE).

$$L_{rank} = \sum_{i} \sum_{s_i > s_j} \max\left[0,\ w - (\hat{s}_i - \hat{s}_j)\right], \tag{23}$$

where w is a margin hyperparameter generally set to 1. The arrangements of ground-truth scores s and predicted scores $\hat{s}$ are expected to be consistent for all architecture pairs in the dataset, corresponding to the minimization of (23).

This performance evaluation strategy's effectiveness is demonstrated by experimental results in Section V.

D. BO-based NAS Scheme in Top-level

Exploration and exploitation should be traded off appropriately in this search space $\mathscr{G}$. Upper Confidence Bound (UCB) function is maximized as the acquisition function in BO.

The NAS scheme is summarized in Algorithm 2. First, neural architectures are randomly generated and trained to obtain the initial data pool. Subsequently, all DAGs are encoded into embeddings, mapping the search space into continuous. Then BO iterations begin, where the performance predictor is used to avoid training candidates in BO. GP model is then built and optimized to produce a candidate. This new architecture is also evaluated via the predictor and merged into the data pool. Finally, the recorded optimal embedding is selected.

Algorithm 2 BO-based NAS for Designing Hotspot Detectors

1: Generate the initial data pool $\{(g_i, s_i)\}$ randomly
2: Encode DAGs into latent vectors and get $\{(g_i, \mathbf{z}_i, s_i)\}$
3: **for** $t \leq iterations$ **do**
4: Train $\mathcal{P}(\cdot)$ based on $\{(\mathbf{z}_i, s_i)\}$ and get $\{(\mathbf{z}_i, \hat{s}_i)\}$
5: Construct GP model based on $\{(\mathbf{z}_i, \hat{s}_i)\}$
6: Generate candidate $\mathbf{z}_t$ by optimizing $\mathrm{UCB}(\mathbf{z})$
7: Calculate the score $\hat{s}_t$ and update dataset $\{(\mathbf{z}_i, \hat{s}_i)\}$
8: **end for**
9: **return** the optimal $\mathbf{z}^*$ recorded

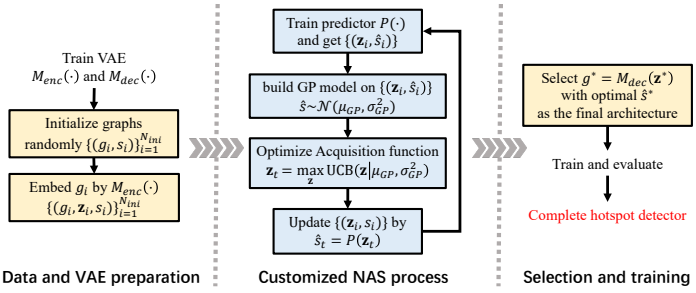


Fig. 4. The proposed scheme's overall workflow.

E. Overall Workflow

The overall workflow is shown in Fig. 4.

1) Data and VAE preparation: The VAE is trained in advance. N_{ini} hotspot detectors are then randomly sampled in the space $\mathscr{G}$. Each hotspot detector g_i is evaluated on the validation set after epoch(s) of training, getting s_i, the ground-truth F_1 score. Finally, g_i is encoded into a vector $\mathbf{z}_i$.

2) Customized NAS process: See Section III-D for details.

3) Selection and training: The latent vector $\mathbf{z}^*$ with the highest $\hat{s}^*$ is selected and decoded into $g^* \in \mathscr{G}$ through $\mathcal{M}_{dec}(\cdot)$. The complete hotspot detector represented by g^* is trained on the training set until the best performance is achieved on the validation set, and taken out as the final hotspot detector.

IV. Methodological Details

There are still additional details to be illustrated, including VAE's training settings and superior performance, the adopted datasets, and training techniques for hotspot detectors.

A. Implementation of VAE

In the VAE's training process, 10^5 DAGs are randomly sampled to generate the dataset $\{(g_i)\}$, $g_i \in \mathscr{G}_{cell}$. The dimension of the Euclidean space $\mathscr{R}^{m/2}$ is set to 14.

Subsequently, network weights are randomly initialized. In each iteration, the encoding-decoding process is performed for

all samples. GD then optimizes the loss function in (11) to update weights, where hyperparameter β is set to 0.01.

Convergence has been reached after 50 epochs of training. The reconstruction accuracy exceeds 99.95% on both the training and validation set divided by the proportion of 9:1.

Besides perfect reconstruction accuracy, the ability to distinguish between different graphs is another crucial advantage of the proposed VAE. The following test is performed. First, graphs g_1 and g_2 are randomly generated; g_1 is randomly perturbed to obtain a similar graph $\widetilde{g}_1$; then the Euclidean norms between their latent vectors are calculated, respectively recorded as $d(g_1, g_2)$ and $d(g_1, \widetilde{g}_1)$. The above operations are repeated 10,000 times to obtain the average distances $\bar{d}(g_1, g_2) = 10.204$ and $\bar{d}(g_1, \widetilde{g}_1) = 2.466$. Therefore, this VAE can indeed measure the difference between graphs.

B. Layout Hotspot Datasets

ICCAD 2012 [13] and ICCAD 2019 [7] Contest benchmarks are used, referred to as ICCAD12 and ICCAD19, respectively. In ICCAD12, more than 97% layouts relate to a $28\mathrm{nm}$ process design kit (PDK), and the remaining relates to a $32\mathrm{nm}$ PDK. ICCAD19 inherits the $28\mathrm{nm}$ PDK and layer types in ICCAD12, updating the lithographic models and adding new patterns for testing. ICCAD19 is divided into two subsets, ICCAD19-1 and ICCAD19-2, with a shared training set.

C. Training of Hotspot Detectors

Several customized techniques are adopted in the hotspot detectors' training process due to the characteristic of layout datasets. As shown in Fig. 3(b), feature extraction is performed on the original layout clips before feeding them into NN-based hotspot detectors. Here, the discrete cosine transform (DCT) [17] is employed to extract features.

In addition, the dataset in Section IV-B is unbalanced. An up-sampling method is adopted during the training process to balance the sampling probability of hotspots and non-hotspots. A biased learning method [17] is introduced to balance detecting accuracy and false alarm rate.

V. Experimental Results

A. Experimental Settings

NAS experiments are conducted on a single NVIDIA RTX 2080Ti GPU with ICCAD12 and ICCAD19 benchmarks. The searched cells for these benchmarks are shown in Fig. 5. The whole scheme is deployed in *Pytorch* and its detailed settings are presented here.

1) Predictor settings: A two-layer MLP is sufficient. Sigmiod function serves as the output activation.

2) Hotspot detectors' training settings: Stochastic GD is utilized for training detectors. The initial learning rate is set to 0.025 with a batch size of 128; the learning rate is annealed down to 0 with the cosine scheduler; Total 8 cells are connected serially to build a macro neural architecture.

3) Top-level NAS settings: The *Botorch* toolkit constructs the GP model and optimizes the UCB function. Total 600 epochs of search are enough to ensure the convergence of BO. Total 100 detectors are sampled and trained for one epoch to initialize the data pool.

B. Comparison with Manual Designs

Table III shows the performance comparison of hotspot detectors auto-designed by the proposed method and manually designed ones. Columns *Accu*, *FA*, F_1 and *Time* show the hotspot detection accuracy, false alarm rate, F_1 score, and hotspot detector's inference time, respectively.

Taking the column *Accu* in the *TCAD'19* [17] subtable as an example, *Average* represents the average of the accuracy obtained by *TCAD'19* [17] on these datasets; and *Ratio* represents the average accuracy's ratio of *TCAD'19* [17] to *Ours*. The paradigms above can be migrated to other subtables.

Specifically, compared with all manually designed hotspot detectors, ones generated by the proposed method achieve the highest average accuracy, the lowest average FA, and the shortest average inference time.

The results above demonstrate that auto-designed hotspot detectors surpass manually designed ones in terms of detection capability and operating efficiency. Moreover, in these baselines, professionals have spent months designing NNs, which auto-designed ones defeat in about a single hour of search. In addition, the used NN models with limited scale (stacking layers) also avoid long inference times.

C. Comparison with Other NAS-based Methods

Table IV compares the proposed scheme with other NAS-based methods. *DARTS* [8] is a classic general-purpose NAS algorithm; *TODAES'22* [10] is the state-of-the-art NAS-based hotspot detection method.

Compared with the general *DARTS* [8] and the customized *TODAES'22* [10], this scheme achieves the best performance in all hotspot detection metrics and reduces the search time by $15.120\times$ and $11.173\times$, respectively.

The results above illustrate that the proposed scheme significantly improves the design efficiency and maintains competitive hotspot detection performance, surpassing the existing NAS-based methods.

On the one hand, although NAS-based methods above share the same topological search space in Section III-A, this paper's VAE maps the search space to a compact continuous space to facilitate design space exploration in BO. Therefore, it is reasonable to obtain hotspot detectors with higher performance finally. On the other hand, the performance predictor reveals the potential correlation between the architecture and its performance, avoiding repeated training, thus significantly improving design efficiency.

VI. Conclusions

This paper proposes a NAS scheme to design high-performance NN-based hotspot detectors agilely and automatically. VAE is conducive to performing BO in continuous space to obtain superior solutions, and performance prediction in the local solution space makes our scheme more agile. Experiments conducted on ICCAD12 and ICCAD19 demonstrate that the proposed scheme significantly increases the agility of auto-design and generates hotspot detectors competitive with state-of-the-art ones in all metrics. NN-based hotspot detectors are therefore promoted to move towards automation and agility.

TABLE III
PERFORMANCE COMPARISON WITH STATE-OF-THE-ART MANUALLY DESIGNED NN-BASED HOTSPOT DETECTORS

Dataset	**TCAD'19** [17]				**DAC'19** [18]				**TCAD'19** [19]				**Ours**			
	Accu (%)	FA (%)	F_1 score	Time (s)	Accu (%)	FA (%)	F_1 score	Time (s)	Accu (%)	FA (%)	F_1 score	Time (s)	Accu (%)	FA (%)	F_1 score	Time (s)
ICCAD12	98.4	26.2	0.581	397	99.2	20.6	0.641	60	97.7	17.9	0.666	401	96.2	6.4	0.838	7.5
ICCAD19-1	76.0	2.6	0.710	53	80.9	3.5	0.667	63	47.0	1.5	0.556	56	91.6	8.6	0.577	8.3
ICCAD19-2	88.4	87.8	0.636	429	89.7	84.1	0.651	496	84.5	83.6	0.627	445	90.5	83.9	0.656	44.9
Average	87.6	38.9	0.642	293	89.9	36.1	0.653	207	76.4	34.3	0.616	301	92.8	33.0	0.690	20.2
Ratio	0.944	1.179	0.930	14.505	0.969	1.094	0.946	10.248	0.823	1.039	0.893	14.901	1	1	1	1

TABLE IV
PERFORMANCE COMPARISON WITH STATE-OF-THE-ART NAS-BASED METHODS

Dataset	**DARTS** [8]					**TODAES'22** [10]					**Ours**				
	Accu (%)	FA (%)	F_1 score	Time (s)	Search time (h)	Accu (%)	FA (%)	F_1 score	Time (s)	Search time (h)	Accu (%)	FA (%)	F_1 score	Time (s)	Search time (h)
ICCAD12	90.4	7.4	0.790	8.1	9.62	93.3	9.5	0.767	8.3	6.81	96.2	6.4	0.838	7.5	0.76
ICCAD19-1	84.5	8.3	0.554	5.9	9.63	90.8	8.6	0.573	7.3	6.78	91.6	8.6	0.577	8.3	0.73
ICCAD19-2	90.0	88.7	0.642	65.6	14.77	85.5	84.4	0.630	49.8	11.56	90.5	83.9	0.656	44.9	0.76
Average	88.3	34.8	0.662	26.5	11.34	89.9	34.2	0.657	21.8	8.38	92.8	33.0	0.690	20.2	0.75
Ratio	0.952	1.055	0.959	1.314	15.120	0.968	1.035	0.952	1.079	11.173	1	1	1	1	1

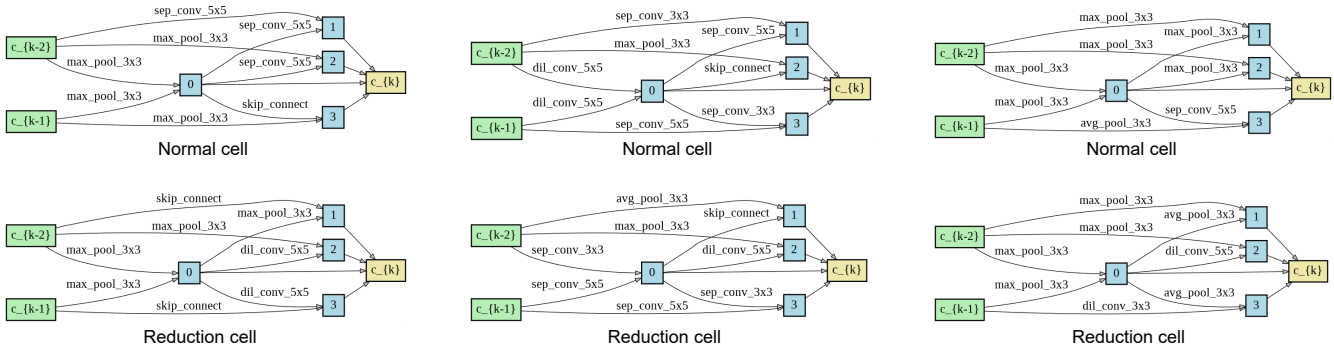


Fig. 5. The searched cells by the proposed scheme. Figures 5(a), 5(b) and 5(c) are cells searched on ICCAD12, ICCAD19-1, and ICCAD19-2 benchmarks. c_{k-2} and c_{k-1} represent two input nodes from two nearest former cells, and c_k represents the only output node.

ACKNOWLEDGEMENT

This research is supported partly by National Key R&D Program of China 2020YFA0711900, 2020YFA0711903, partly by National Natural Science Foundation of China (NSFC) research projects 62141407, 61929102 and 62090025.

REFERENCES

[1] Ibrahim M Elfadel, Duane S Boning, and Xin Li. *Machine learning in VLSI computer-aided design*. Springer, 2019.

[2] Andrew B Kahng, Chul-Hong Park, and Xu Xu. Fast dual graph-based hotspot detection. In *Photomask Technology 2006*, volume 6349, page 63490H. International Society for Optics and Photonics, 2006.

[3] Yen-Ting Yu, Ya-Chung Chan, Subarna Sinha, Iris Hui-Ru Jiang, and Charles Chiang. Accurate process-hotspot detection using critical design rule extraction. In *Proceedings of the 49th Annual Design Automation Conference*, pages 1167–1172, 2012.

[4] Dragoljub Gagi Drmanac, Frank Liu, and Li-C Wang. Predicting variability in nanoscale lithography processes. In *2009 46th ACM/IEEE Design Automation Conference*, pages 545–550. IEEE, 2009.

[5] Haoyu Yang, Jing Su, Yi Zou, Yuzhe Ma, Bei Yu, and Evangeline FY Young. Layout hotspot detection with feature tensor generation and deep biased learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(6):1175–1187, 2018.

[6] Yiyang Jiang, Fan Yang, Bei Yu, Dian Zhou, and Xuan Zeng. Efficient layout hotspot detection via binarized residual neural network ensemble. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(7):1476–1488, 2020.

[7] Gaurav Rajavendra Reddy, Kareem Madkour, and Yiorgos Makris. Machine learning-based hotspot detection: Fallacies, pitfalls and marching orders. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[8] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[9] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. *Advances in neural information processing systems*, 31, 2018.

[10] Yiyang Jiang, Fan Yang, Bei Yu, Dian Zhou, and Xuan Zeng. Efficient layout hotspot detection via neural architecture search. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2022.

[11] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. *Advances in Neural Information Processing Systems*, 32, 2019.

[12] Xuefei Ning, Wenshuo Li, Zixuan Zhou, Tianchen Zhao, Shuang Liang, Yin Zheng, Huazhong Yang, and Yu Wang. A surgery of the neural architecture evaluators. 2020.

[13] J Andres Torres. Iccad-2012 cad contest in fuzzy pattern matching for physical verification and benchmark suite. In *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 349–350. IEEE, 2012.

[14] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

[15] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[16] Jialin Lu, Liangbo Lei, Fan Yang, Li Shang, and Xuan Zeng. Topology optimization of operational amplifier in continuous space via graph embedding. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 142–147. IEEE, 2022.

[17] Haoyu Yang, Jing Su, Yi Zou, Yuzhe Ma, Bei Yu, and Evangeline FY Young. Layout hotspot detection with feature tensor generation and deep biased learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(6):1175–1187, 2018.

[18] Yiyang Jiang, Fan Yang, Hengliang Zhu, Bei Yu, Dian Zhou, and Xuan Zeng. Efficient layout hotspot detection via binarized residual neural network. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[19] Ying Chen, Yibo Lin, Tianyang Gai, Yajuan Su, Yayi Wei, and David Z Pan. Semisupervised hotspot detection with self-paced multitask learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(7):1511–1523, 2019.