DeePattern: Layout Pattern Generation With Transforming Convolutional Auto-Encoder

Haoyu Yang, Shuhe Li, Wen Chen, Piyush Pathak, Frank Gennari, Ya-Chieh Lai, and Bei Yu, *Member, IEEE*

***Abstract*—VLSI layout patterns provide critical resources in various design for manufacturability research, from early technology node development to back-end design and sign-off flows. However, a diverse layout pattern library is not always available due to long logic-to-chip design cycle, which slows down the technology node development procedure. To address this issue, in this paper, we explore the capability of generative machine learning models to synthesize layout patterns. A transforming convolutional auto-encoder (TCAE) family is developed to learn vector-based instantiation of squish pattern topologies. We show our TCAE models can capture simple design rules and enlarge the existing squish topology space under certain transformations. With adaptive configurations, the proposed G-TCAE framework allows both massive pattern generation and context-specific pattern generation. Geometry information of each squish topology is obtained from an associated linear system derived from design rule constraints. Experiments on 7*nm* EUV designs show that our framework can more effectively generate diverse pattern libraries with DRC-clean patterns compared to a state-of-the-art industrial layout pattern generator.**

***Index Terms*—Lithography, design for manufacture, machine learning.**

I. Introduction

VLSI layout patterns provide critical resources in a variety of design for manufacturability (DFM) researches, from (1) early technology node development [1] to (2) back-end design and sign-off flows [2]. The former includes perfection of design rules [3], OPC recipes [2], [4], lithography models [5]–[7] and so on. The latter covers, but is not limited to, layout hotspot detection and correction [8]–[16]. However, layout pattern libraries are sometimes not large and diverse enough for DFM research/solutions due to long logic-to-chip design cycle. Even some test layouts can be synthesized within a short period, they are usually restricted by certain design rules and the generated pattern diversity is limited [5], [17], [18].

One state-of-the-art industrial pattern generation solution is migrating designs from older technology node [19], where seed patterns are selected from the process weak points in older designs and shrunk by a scale factor to match current design rules. However, such flow is no longer applicable when there are large gaps between design nodes. An example is some 7*nm* EUV layers contain all unidirectional shapes while patterns are still 2D in previous design nodes [20]. Monte Carlo shape generation is another approach that is typically applied industry-wide to generate metal layer patterns for lithography and OPC research. In this method, shapes are randomly created and placed on a given region according to certain geometry constraints, which, to some extent, limits the pattern library diversity created with the flow (see Fig. 1(a)) [21]. Recently, [6] propsoed a GAN approach to generate DCT signals that can be converted back to layout space for machine learning-based OPC model training and resist model calibration. Reference [22] is another attempt using machine learning model for test layout generation, where a set of auto-encoders are trained to produce ISP map and hence practical layouts. There are also random pattern generation solutions with legacy algorithms like VIPER [1] using random walk and [23] with Monte Carlo approach.

In this paper, we study the capability of a new family of generative models to synthesize test layout patterns. For simplicity, in this work we only focus on uni-directional EUV layers in our experiments. It should be noted that the precondition of uni-direction patterns seemingly makes the pattern generation task trivial with previous solutions. However, the goal of context specific pattern generation poses extreme challenges on legacy solutions, which therefore motivates us to design certain flow for such purposes. The most popular generative machine learning model recently is generative adversarial network (GAN) that was first proposed in [24]. GAN consists of two sub-neural networks that interact with each other. One is called generator, which takes random latent vectors as inputs and generates images or patterns following some distribution as of the training data set. The other one is called discriminator, which aims to discriminate real patterns from the generated ones. GAN models have been widely studied on computer vision related tasks like image synthesis [24], [25] and scene transformation [26], [27]. In EDA area, such architecture and its variations have also been successfully applied in lithography simulation [28], mask optimization [29]–[31] and on-chip sensor placement [32]. However, generating layout patterns is

Manuscript received October 24, 2021; revised November 29, 2021; accepted December 20, 2021. Date of publication December 30, 2021; date of current version February 3, 2022. This work was supported in part by the Cadence Design Systems and in part by the Research Grants Council of Hong Kong SAR under Project CUHK24209017. The preliminary version has been presented at the ACM/IEEE Design Automation Conference (DAC), 2019, doi: 10.1145/3316781.3317795. *(Corresponding author: Haoyu Yang.)*

Haoyu Yang is with ASIC and VLSI Research, NVIDIA Corporation, Austin, TX 78717 USA (e-mail: phdyang007@gmail.com).

Shuhe Li, Wen Chen, and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR.

Piyush Pathak, Frank Gennari, and Ya-Chieh Lai are with Design and Sign-Off Group, Cadence Design Systems, Inc., San Jose, CA 95134 USA.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TSM.2021.3139354.

Digital Object Identifier 10.1109/TSM.2021.3139354

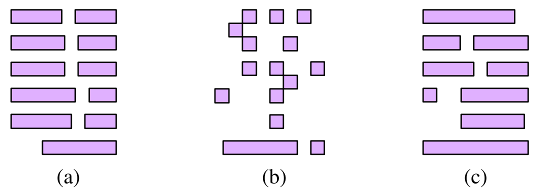


Fig. 1. Sample pattern topologies generated from (a) Industry Tool, (b) DCGAN [36], and (c) our proposed TCAE.

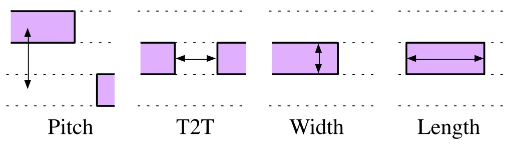


Fig. 2. Layout geometry and critical dimensions.

a more challenging task as layouts are constrained by design rules which are hard to learn with native GAN training mechanisms [33]–[35], as shown in Fig. 1(b). For the example of 7*nm* EUV designs, M2 layer shapes are all unidirectional and cannot appear in adjacent wire tracks.

An alternative approach is the transforming auto-encoders (TAEs) which are designed to automatically learn features that instantiate image objects with variations in position, orientation, scale and so on [37]. TAEs are a group of densely connected auto-encoders and each individual, referred as capsule, is targeting on certain transformations. Such architecture agrees with the pattern generation tasks in the following aspects. (1) Feature instantiation attains data set domain properties. (2) All capsules contribute together to produce variations of any input objects. However, TAEs cannot be directly applied here due to the fact that transformations are restricted by layout design rules and variations of generated patterns are limited by the number and size of capsules which is highly correlated with computational cost.

To overcome these problems, we propose a transforming convolutional auto-encoder (TCAE) architecture that consists of a recognition unit and a generation unit. The recognition unit is built with stacked convolutional layers that convert layout pattern topologies into latent vectors that allow perturbation for certain transformations. The generation unit is expected to capture layout spatial information (e.g., track and wire direction) well and converts latent vectors back to legal pattern topologies with corresponding deconvolution operations. We will show that perturbations on individual latent vector node contributes to pattern transformation in layout domain and admits a larger pattern library diversity, as shown in Fig. 1(c) The architecture is also expected to be computationally efficient. Another advantage of TCAE is that transformations are introduced in the inference stage that brings more room to manipulate features for the perspective of pattern diversity, while transformations in TAEs are pre-assigned during training. Additionally, we construct the hybrid GAN-guided TCAE (G-TCAE) to allow **(1) massive pattern generation** and **(2) context-specific pattern generation** with certain training and inferencing configurations. The main contributions of this paper are listed as follows.

- We propose a pattern generation framework that reduces the challenging pattern generation problem into two simpler subproblems with the aid of efficient squish pattern representation.
- We design a TCAE architecture that aims at generating pattern topologies efficiently.
- We introduce the GAN-guided TCAE that targets at massive DRC-clean pattern generation (following certain design rules) without losing pattern library diversity; G-TCAE also enables generation of content-specific patterns.
- We develop linear systems that can provide solutions of pattern geometric information which, combined with generated squish topology, produces DRC-clean layouts.
- We conduct experiments to show that each latent vector node in TCAE has real physical meanings in layout domain and transformations on latent vectors can produce additional topologies of interest. Results show that the generated pattern library exhibits larger pattern number and pattern diversity compared to the state-of-the-art industry layout generator.

The rest of the paper is organized as follows. Section II introduces basic terminologies and evaluation metrics related to this work. Section III presents details of the pattern generation framework including TCAE-based topology generation and legal pattern assessment. Section IV covers experimental results and Section V concludes the paper.

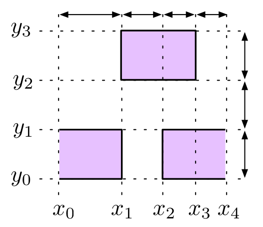
II. Preliminaries

In this section, we introduce the geometry concepts to accommodate layout design rules and the pattern generation flow. Because in this paper we focus on 7*nm* EUV M2 layer designs which contain only unidirectional on-track shapes, following terms are accordingly adopted to describe design rules as depicted in Fig. 2. *Pitch*, denoted as p, measures the distance between two adjacent tracks that contain shapes. *T2T*, denoted as t, measures the line-end-to-line-end distance between two adjacent shapes in a track. Wire *length* l and *width* w measure the shape size along and against the design track, respectively. In order for a pattern to be DRC-clean, these measurements will be constrained by some design rules.

All shape edges in a fixed-size window are aligned with x-axis and y-axis. If we extend all horizontal and vertical edges infinitely into scan lines, the window will be cut into rectangle grids and more non-overlapping scan lines imply higher pattern complexity. We hence define the complexity of a layout pattern as follows.

Definition 1 (Pattern Complexity): The complexity of a pattern in x and y directions (denoted as c_x and c_y) is defined as the number of scan lines subtracted by one along x-axis and y-axis, respectively.

We also introduce the concept of *pattern diversity* (denoted as H) to measure how are the pattern complexities distributed in a given library. A larger H implies the library contains patterns that are more evenly distributed, as in the following definition.



$$\boldsymbol{T} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

$$\boldsymbol{\delta}_x = \begin{bmatrix} x_1 - x_0 & x_2 - x_1 & x_3 - x_2 & x_4 - x_3 \end{bmatrix}$$
$$\boldsymbol{\delta}_y = \begin{bmatrix} y_1 - y_0 & y_2 - y_1 & y_3 - y_2 \end{bmatrix}$$

Fig. 3. Squish pattern generation.

Definition 2 (Pattern Diversity): The diversity of a pattern library is given by the *Shannon Entropy* [38] of the pattern complexity sampled from the library, as shown in Equation (1),

$$\mathrm{H} = -\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} P\big(c_{x,i}, c_{y,j}\big)\log P\big(c_{x,i}, c_{y,j}\big), \tag{1}$$

where $P(c_{x,i}, c_{y,j})$ is the probability of a pattern sampled from the library has complexities of $c_{x,i}$ and $c_{y,j}$ in x and y directions respectively, m and n are the total number of different complexities along x and y-axes respectively.

With above definitions, the pattern generation problem can be formulated as follows.

Problem 1 (Pattern Generation): Given a set of layout design rules, the objective of pattern generation is to generate a pattern library such that the pattern diversity and the number of unique DRC-clean patterns in the library is maximized.

III. Pattern Generation With TCAE Family

Generating layout patterns is extremely challenging for learning machines as design rules are usually not friendly to most machine learning models. We therefore simplify the problem into two stages with the aid of squish patterns [39]. We first deal with the topology generation problem and then we establish a linear system to finalize the pattern generation flow with proper geometry vectors.

A. Squish Pattern Extraction

Squish pattern is a scan line-based representation that each layout clip is cut into grids aligned at all shape edges, as shown in Fig. 3. The squish pattern representation of a given layout clip consists of a topology matrix $\boldsymbol{T}$ and two vectors $\boldsymbol{\delta}_x$ and $\boldsymbol{\delta}_y$ that contain geometry information in x and y directions. Each entry of $\boldsymbol{T}$ is either zero or one which indicates space or shape respectively. The geometric information describes the size of each grid. For example, the pattern in Fig. 3 can be accordingly expressed as in the right side of Fig. 3. Here x_is and y_is are the locations of vertical and horizontal scan lines respectively, and the pattern complexity is accordingly given by $c_x = 4$ and $c_y = 3$. Canonically, (x_0, y_0) is the coordinate of the bottom left corner of the pattern and $x_i < x_{i+1}, y_i < y_{i+1}$. From the squish pattern extraction procedure, we can observe that *squish pattern representation is lossless.*

Now the problem becomes generating legal topologies and solving associated $\boldsymbol{\delta}_x$s and $\boldsymbol{\delta}_y$s that are much easier than directly generating DRC-clean patterns with image-based generative learning models. Most of the generative machine learning models perform pixel-wise image generation that follows some distribution. If we directly generate layout images with these models, it will be quite hard to create valid polygons or rectangles. Because generated shape pixels could appear anywhere in the image matrix. Also, design rules cannot be easily handled within these image generation models. However, with the help of squish patterns, the layout image generation task is split into two simpler phases and now we can guarantee that generated patterns are always polygons or rectangles as desired. The advantages of squish patterns are two-fold:

1) Squish patterns are storage-efficient compared to pixel-based layout images. Given a specific layout, squish patterns are more likely to present the same information as pixel-based layout images (e.g., $1nm^2$/pixel) with much smaller topology matrix and geometry vectors. For example, suppose the layout in Fig. 3 is with $64nm \times 64nm$. For image-based representation, we need 64×64bits $=$ 512bytes to store the layout. However with squish patterns, we only need 1.5bytes (topology matrix)$+7 \times 4$bytes (geometric vectors) $=$ 29.5bytes storage. Although GDSII or OASIS can store layouts more efficiently, these formats are not compatible with deep learning models.
2) Squish patterns are naturally compatible with the neural network-based pattern generation flow that will be discussed in following sections.

B. TCAE Basis

1) Transforming Convolutional Auto-Encoder (TCAE): In this paper, we propose a TCAE architecture that aims at efficient pattern topology $\boldsymbol{T}$ generation. The TCAE is inspired from original transforming auto-encoders (TAEs) [37] which are a group of densely connected auto-encoders and each individual, referred as a capsule, is targeting on certain image-to-image transformations. Each capsule employs a recognition unit and a generation unit to capture pose positions and re-synthesize the translated object, respectively. The translation (e.g., shift or rotation) is defined on a regular coordination system that will be added up to the original pose position before fed into the generation unit. In the training phase, neuron weights are updated by backpropagating the differences between the output image and the actual translated image given the translation information. After the neural network is trained, it takes inputs of an image and translation information and outputs the image with desired shifts. Such architecture agrees with the pattern generation tasks in the following aspects. (1) Feature instantiation attains data set domain properties. (2) All capsules contribute together to produce variations of any input objects. Although such architecture can capture the feature characteristics from original object pixel intensities, TAEs cannot be directly applied for pattern generation due to the fact that *transformations are restricted by layout design rules and only very simple pose transformations are supported by original TAEs, which does not satisfy our pattern generation objectives.*

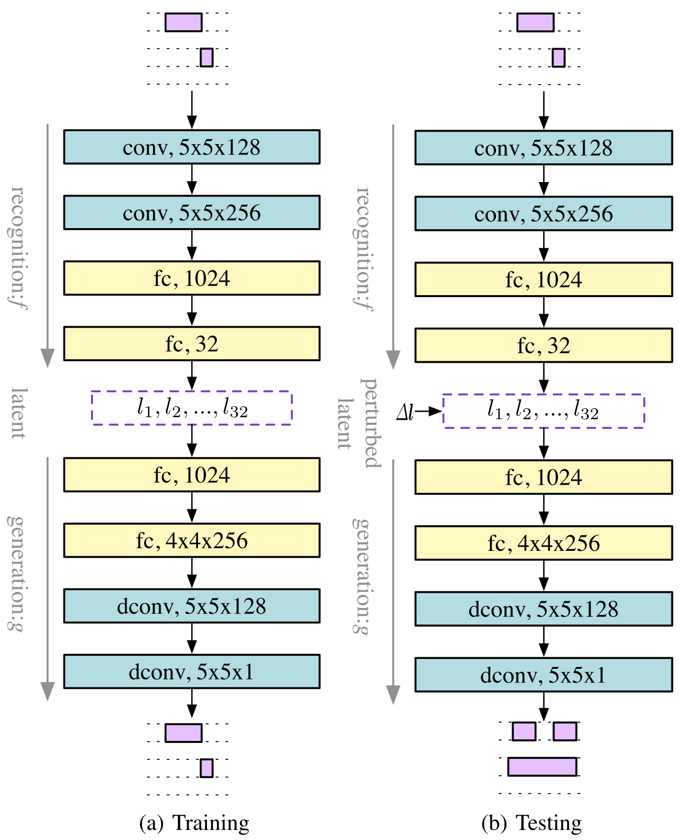


Fig. 4. Architecture of transforming convolutional auto-encoder in (a) training phase and (b) testing phase. conv, fc and dconv represent convolution layer, fully connected layer and deconvolution layer respectively.

We develop the TCAE architecture for feature learning and pattern reconstruction, as shown in Fig. 4. The detection unit in TCAE consists of multiple convolutional layers for hierarchical feature extraction, followed by several densely connected layers as an instantiation of the input pattern in the latent vector space, as in Equation (2).

$$\boldsymbol{l} = f(\boldsymbol{T}; \boldsymbol{W}_f), \tag{2}$$

where $\boldsymbol{l}$ is the latent vector, $\boldsymbol{T}$ represents the input topology and $\boldsymbol{W}_f$ contains all the trainable parameters associated with the recognition unit. The latent vector works similarly as a group of capsule units with each node being an low level feature representation. We will show each latent vector node contributes to pattern shape globally or locally in the experiment section.

The generation unit contains deconvolution layers [40] that cast the pattern object from the latent vector space back to the original pattern space, as in Equation (3).

$$\boldsymbol{T}' = g(\boldsymbol{l} + \Delta\boldsymbol{l}; \boldsymbol{W}_g), \tag{3}$$

where $\Delta\boldsymbol{l}$ is the perturbation applied on the latent vector that performs transformations on inputs. During the training phase, we force the TCAE to learn an identity mapping with the following objectives.

$$\min_{\boldsymbol{W}_f, \boldsymbol{W}_g} ||\boldsymbol{T} - \boldsymbol{T}'||_2, \text{ s.t. } \Delta\boldsymbol{l} = \boldsymbol{0}. \tag{4}$$

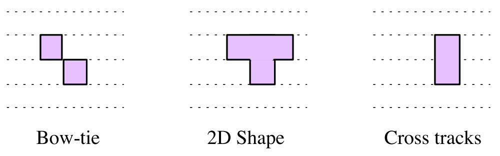


Fig. 5. Illegal topology examples.

The TCAE-based framework differs from TAEs in the following aspects.

- We replace the group capsules with a simpler latent vector that contains feature nodes connected to different receptive fields in early convolution layers and hence can represent certain part-whole feature instantiation.
- The transformation in our framework will be applied directly on the latent vector space that promises a much larger diversity of the generated patterns compared to the limited transformation on the coordinate system in TAEs.
- Identity mapping in the training phase helps the TCAE capture the design rules of existing patterns.

Once the TCAE is trained, we can adopt the flow in Fig. 4(b) to generate pattern topologies from perturbed latent vector space of existing layout patterns. During the inference phase, we feed a group of squish topologies into the trained recognition unit that extract latent vector instantiations of existing topologies. Perturbation on the latent vector space is expected to expand the existing pattern library with legal topologies. Although there is no mathematical guarantee that a trained auto-encoder is able to produce diverse outputs with perturbations in the bottleneck layer, we can still expect the proposed flow to be a sound solution. According to Equations (2) and (3), the trained f can be viewed as a mapping (one-one mapping if no ReLU is applied) from topology space $\mathcal{T}$ to latent space $\mathcal{L}$. Similarly, g goes the opposite way. Let $\mathcal{T}$ be the topology space given by the existing pattern library, and $\mathcal{L}$ be the corresponding latent space. Hence, the first challenge of the TCAE flow for new pattern generation is to maximize $\Pr(\boldsymbol{T}_n \notin \mathcal{T})$, where

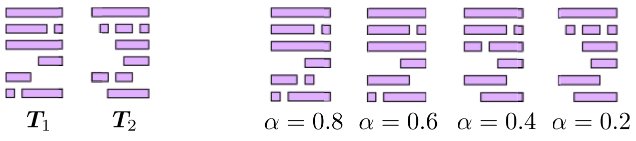
$$\boldsymbol{T}_n = g(\boldsymbol{l}_n; \boldsymbol{W}_g), \boldsymbol{l}_n \notin \mathcal{L}. \tag{5}$$

Because the only non-linearity of $g(f(\bullet))$ comes from ReLU, we can assume $\Pr(\boldsymbol{T}_n \notin \mathcal{T})$ to be large enough (if no ReLU is used, f and g will be strictly linear), and this motivates us to apply noise perturbations for new pattern generation. We claim the following assumption.

Assumption 1: The latent vector space offers richer information and each latent vector node represents patten geometry locally or globally, e.g., some vector nodes may control shape sizes while some others may control shape positions.

2) TCAE-Combine: A straightforward perturbation approach is combining existing topologies in the latent vector space, which is expected to fill empty spaces of a given pattern library. The combination rule can be defined as Equation (6).

$$\boldsymbol{T}_g = g\left(\sum_i \alpha_i f(\boldsymbol{T}_i)\right), \tag{6}$$



(a) Origin (b) $g(\alpha f(\boldsymbol{T}_1) + (1-\alpha) f(\boldsymbol{T}_2))$

Fig. 6. Combination of existing patterns with latent vectors.

where $0 < \alpha_i < 1, \forall i$ are combination coefficients and satisfy $\sum_i \alpha_i = 1$. However, the diversity topologies generated by such approach might be limited by the existing pattern complexity. We randomly pick two patterns from existing designs and combine them in the latent vector space using TCAE. As shown in Fig. 6, even we adjust the combination coefficient with a large step, there are still repeating topologies in the reconstructed topology set.

3) TCAE-Random: Another approach is introducing random perturbations on the latent vectors. To avoid illegal topologies, we take Assumption 1 into consideration and introduce the concept of feature sensitivity $\boldsymbol{s}$ which statistically defines how easily a legal topology can be transformed to an illegal one when manipulating the latent vector node with everything else unchanged.

Definition 3 (Feature Sensitivity): Let $\boldsymbol{l} = \begin{bmatrix} l_1 & l_2 & \cdots & l_n \end{bmatrix}^\top$ be the output of the layer associated with the latent vector space. The sensitivity s_i of a latent vector node l_i is defined as the probability of reconstructed pattern being invalid when a perturbation $\Delta l_i \in [-t, t]$ is added up on l_i with everything else unchanged.

It can be seen from Definition 3 that a larger s_i indicates the corresponding latent vector node l_i is more likely to create invalid topologies if a large perturbation is applied. We therefore avoid manipulating such nodes when sampling random perturbation vectors from a Gaussian distribution. The s_is are estimated following Algorithm. 1, which requires a set of legal topologies and trained TCAE. The sensitivity of each latent vector node is estimated individually (lines 1–2). We first obtain the latent vectors of all topologies in $\mathcal{T}$ and feed them into the reconstruction unit along with certain perturbation on one latent vector node (lines 3–5). Reconstructed patterns are appended in the corresponding set $\mathcal{R}_i$ (line 6). The sensitivity of the latent vector node i is given by the fraction of invalid topologies in $\mathcal{R}_i$ (line 8).

After we get the estimated sensitivity of all latent vector nodes, we are able to sample perturbation vectors whose elements are sampled independently from $\mathcal{N}(0, \frac{1}{s_i})$. These perturbation vectors will be added up to the latent vectors of existing pattern topologies to formulate perturbed latent vectors which will be fed into the generation unit to construct new topologies. Because we focus on EUV metal layers in this work, a topology is illegal if and only if it contains any patterns in Fig. 5. That is, illegal topologies can be filtered out by checking whether shapes appear at any two adjacent tracks.

4) Why Not Pure Variational Auto-Encoders: The encoder-decoder architecture for data generation will possibly remind readers a very similar generative model called variational auto-encoder (VAE) [41], which shares almost the same architecture as TCAE. A VAE also consists of an encoder (recognition unit) and a decoder (reconstruction unit) which are trained with the following loss function:

$$\mathcal{L}(\boldsymbol{W}_e, \boldsymbol{W}_d) = -\mathbb{E}_{z \sim q_{\boldsymbol{W}_d}(z|\boldsymbol{x})}\big(\log p_{\boldsymbol{W}_e}(\boldsymbol{x}|z)\big) + \mathbb{KL}\big(q_{\boldsymbol{W}_d}(z|\boldsymbol{x}) || p(z)\big), \tag{7}$$

where the first term aims to train the neural network to generate a latent vector z that can be reconstructed into data instance following the distribution of the training dataset, and the second term makes z follow a distribution $p(z)$ via K-L Divergence. Usually, $p(z)$ is specified as normal distribution $\mathcal{N}(0, 1)$. In the inference stage, the decoder can create data instances following the training set by taking inputs of $z \sim p(z)$. Equation (7) tells us that a VAE focuses on the generation of some data that follows certain (known) distribution, which shares the same beneath idea as GANs. However, in the task of test layout pattern generation, we expect to generate patterns that do not exist before, which explains why VAE series are discarded as a candidate solution. Although VAE, like GAN, itself does not comply with our pattern generation objectives, it can still contribute to perturbation generation, as will be discussed in the following section.

Algorithm 1 Estimating Feature Sensitivity. $\mathcal{T} = \{\boldsymbol{T}_1, \boldsymbol{T}_2, \ldots, \boldsymbol{T}_N\}$ Is a Set of Valid Pattern Topologies, n is the Number of Nodes in the Latent Layer, f and g Are Trained Recognition Unit and Generation Unit Respectively, t Determines the Perturbation Range, and $\boldsymbol{s}$ Is the Estimated Feature Sensitivity

```
Require: $\mathcal{T}, f, g, t$.
Ensure: $\boldsymbol{s}$.
 1: $\mathcal{R}_i \leftarrow \emptyset, \forall i = 1, 2, \ldots, N$;
 2: for $i = 1, 2, \ldots, n$ do
 3:     for $\lambda = -t : t$ do
 4:         $\Delta \boldsymbol{l} \leftarrow \boldsymbol{0}$, $\Delta l_i \leftarrow \lambda$;
 5:         $\mathcal{T}_i \leftarrow g(f(\mathcal{T}) + \Delta \boldsymbol{l})$;
 6:         $\mathcal{R}_i \leftarrow \mathcal{R}_i + \mathcal{T}_i$;
 7:     end for
 8:     $s_i \leftarrow$ fraction of invalid topologies in $\mathcal{R}_i$;
 9: end for
10: return $\boldsymbol{s}$.
```

C. GAN-Guided TCAE

Latent vector sensitivity analysis allows effective random perturbation by generating more layouts that follow design rules. We avoid to touch sensitive latent nodes during perturbation for the sake of a larger fraction of legal patterns. However, such mechanism exhibits the following drawbacks.

- *Low legal pattern fraction:* Analysis of individual latent nodes does not necessarily guarantee legal pattern generation because multiple entries in a latent vector are more likely to work together and contribute to legal or non-legal patterns. As will seen in the experiments, $< 30\%$ of the patterns in the total generated pattern pool are legal with sensitivity-aware perturbation.

- *Limited controllability of pattern complexity:* Sensitivity-aware perturbations in the latent space only allow generation of legal patterns without control of pattern style and complexity, which narrows the application scope of the pattern generation flow.

To address the above concerns, in this section we introduce an improved TCAE framework which is guided by generative neural networks.

1) Latent Generation With GAN: The working mechanism of TCAE has enabled a mapping between the latent vector space $\mathcal{L}$ and the generated pattern space $\mathcal{T}$. Particularly, for a trained generation unit $g(\cdot; \boldsymbol{W}_g)$ there exists a latent vector set $\mathcal{L}_{\text{valid}}$ that generates valid patterns, i.e.,

$$\mathcal{T}_{\text{valid}} = g\big(\mathcal{L}_{\text{valid}}; \boldsymbol{W}_g\big). \tag{8}$$

Equation (8) implies a high valid pattern fraction will be tightly associated with $\mathcal{L}_{\text{valid}}$. Recall that a generative adversarial networks (GAN) is designed to learn a mapping from some distribution p_z to a target distribution p_{data}, where

$$\boldsymbol{x} = g_{\text{GAN}}\big(z; \boldsymbol{W}_{g-\text{GAN}}\big), \tag{9}$$

where $z \sim p_z$ and $\boldsymbol{W}_{g-\text{GAN}}$ denotes the trained parameter in a GAN generator. Also, if a GAN is trained to reach equilibrium, we will have $\boldsymbol{x} \sim p_{\text{data}}$, as proved in [24, Th. 1]. Let p_{data} to be the distribution of $\mathcal{L}_{\text{valid}}$, we can train a generator that generates latent vectors for valid pattern generation. Accordingly, we design a GAN-guided TCAE (G-TCAE) which contains a hybrid architecture of a generator a GAN and a TCAE, as shown in Fig. 7. In the training phase, a TCAE is trained following Section III, which is able to create massive patterns afterwards. The latent vectors and the results of pattern validity assessments hence provide training sources for GAN. The inference stage requires the interaction between the trained TCAE and GAN. Some random vector drawn from a distribution is fed into the generator, which will generate latent vector that can be directly fed into the generation unit of the TCAE for final pattern generation.

The above discussion provides an insight that a GAN structure offers two main advantages over TCAE: (1) A GAN generates perturbation vectors that follow the distribution of $\mathcal{L}_{\text{valid}}$ which promises a higher fraction of valid patterns in the generated library. (2) The distribution control ability of GAN also allows in demand generation of patterns, such as requirements of certain density, complexity and pattern contexts, as we will show in the experiments. (3) The generation of latent vectors are much easier than direct pattern generation, which makes the framework show much better performance than pure GAN-based layout generation.

2) G-TCAE Flows: Although G-TCAE ideally offers advantages over TCAE, a fit architecture is always inevitable to be part of the aspects that lead to these benefits. The discussion above implies the target of the generator is to provide both (1) perturbations on latent vectors and (2) pure latent vectors. Perturbations on latent vectors target to generate a larger fraction of unique DRC-clean patterns and pure latent vectors are used for topology control. We hence build a hybrid structure that support both, as shown in Fig. 7.

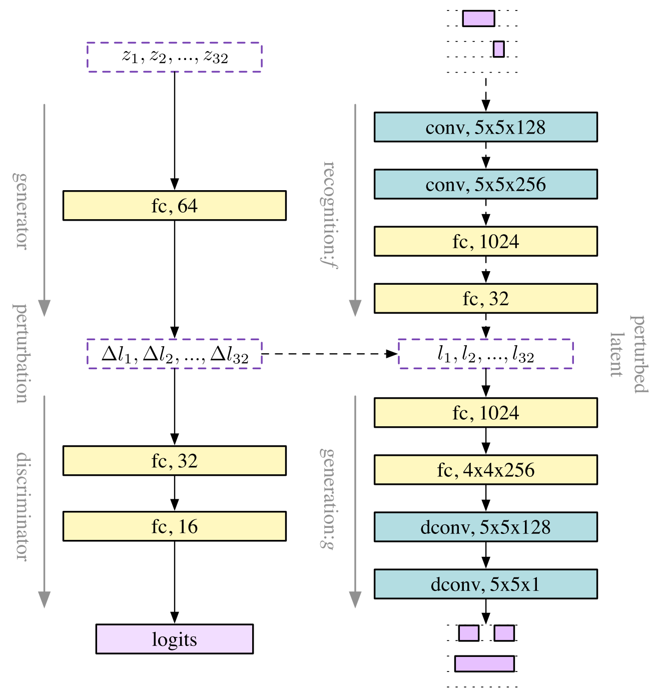


Fig. 7. Hybrid G-TCAE Architecture.

The proposed G-TCAE architecture consists of a standard generative adversarial networks and a TCAE as in Section III. The generator is a standard three-layer perceptron with 64 hidden nodes using Leaky-ReLU activation and batch normalization. The discriminator is designed with two hidden layers for feature transformation and logits prediction. We make the generator shallow and simple for the following reasons: (1) the GAN is only used to generate a latent vector of length 32, thus the vanilla multi-layer perceptron is already powerful enough to complete the task; (2) the outputs of the generator are directly fed into the reconstruction unit for pattern generation, and a complicated generator design is more likely to limit the solution space of generated vectors and hence the diversity of reconstructed patterns.

Either massive pattern generation or context-specific pattern generation requires a well-trained TCAE. Thus in the training procedure, the first step is to train the original TCAE with objectives defined in Equation (4). Depending on different applications of G-TCAE, the GAN part should be trained with different datasets. For the massive unique pattern generation, the training set consists of perturbation vectors that used in TCAE phase which will generate DRC-clean topologies. For the context-specific pattern generation, the training set will be pure latent vectors that reconstruct certain pattern types. These vectors can be obtained by feeding desired types of patterns into the recognition unit of TCAE. It should be noted that these vectors (as real data) will also be fed into the discriminator together with the generator-generated vectors (as faked data) to complete the GAN training. The training procedure of GAN is exactly the same as [24], where neuron weights in the generator and the discriminator are updated alternatively. After G-TCAE is trained, the discriminator is no longer taking effect.

Similarly, there are also different testing flows for two application scenarios. In massive pattern generation, the computing graph contains the generator, the recognition unit and the reconstruction unit. The recognition unit will take existing topology as inputs and produce their corresponding latent vectors $\boldsymbol{l}$. The generator takes some random vectors as inputs and generate perturbation noises $\Delta\boldsymbol{l}$. The reconstruction unit will take the perturbed latent vectors $\boldsymbol{l} + \Delta\boldsymbol{l}$ as inputs and generate new patterns. In context-specific pattern generation, the recognition unit is discarded, and the generator will directly generate latent vectors for the generation unit.

3) Discussion: One thing we would like to mention is the poor performance of standard GAN and VAE in the pattern generation task. Superficially, GAN and VAE are designed to generate instances that follow certain distributions (the distribution of the training set), which seems to be applicable to the pattern generation problem in this paper. However, due to the potential insufficiency or weak distribution of the training data set, GAN and VAE would be easily collapsed with poor generation ability. We will show the supporting results later in Section IV.

The TCAE framework, on the other hand, uses the original auto-encoder as the backbone architecture and tries to converts the layout into latent space. At the AE training stage, we do not learn any distributions from the training set and we only make sure that a good latent vector will always be converted to a DRC-clean topology. To expand the layout library in terms of unique pattern count and pattern diversity, we place random noise to perturb the good latent vectors. Obviously, these noise will definitely make the latent vectors not good anymore (i.e., will generate invalid topology after decoding). This problem is still solvable. In our experiments, we observe that different latent vector nodes correspond to different topology characteristics (line-end, spacing or direction). We therefore propose the sensitivity analysis of the latent vector nodes. *With this approach, we can find out latent nodes that will easily result in invalid topology if perturbed with large noises.* And in the perturbation stage we will avoid to make large perturbations on these nodes.

Although sensitive analysis on latent vectors will to some extent avoid invalid topology generation, this approach does not consider the interaction among various number of latent nodes. To improve the valid topology generation efficiency, we therefore propose the architecture of generative model enhanced TCAE (GAN-TCAE as our case study). We hope the generative model can be applied here to directly generate latent vectors that will be decoded as valid topologies. Another motivation of the GAN-TCAE architecture is the context-specific pattern generation that controls pattern complexities and styles, and this is almost impossible with only random enumeration on pure TCAE.

D. Legal Pattern Assessment

To generate DRC clean patterns, we need legal $\boldsymbol{\delta}_x$s and $\boldsymbol{\delta}_y$s of all generated topologies. We first detect all critical dimensions listed in Fig. 2 in each valid topology and then formulate a linear system combining all associated constraints, as in

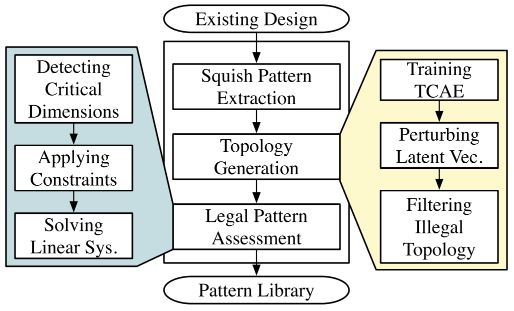


Fig. 8. TCAE flow.

Equation (10).

$$y_{i+1} - y_i = \frac{p}{2}, \qquad \forall i, \tag{10a}$$
$$x_i - x_j = t_{\min}, \qquad \forall (i,j) \in \mathcal{C}_{T2T}, \tag{10b}$$
$$x_i - x_j = l_{\min}, \qquad \forall (i,j) \in \mathcal{C}_{W}, \tag{10c}$$
$$x_{i+1} - x_i > 0, \qquad \forall i, \tag{10d}$$
$$x_{\max} - x_0 = d_x, y_{\max} - y_0 = d_y \tag{10e}$$

where d_x, d_y, $t_{\min}$ and $l_{\min}$ denote clip width, clip height, minimum tip-to-tip distance and wire length (refer to Fig. 2), respectively. Each index pair $(i,j) \in \mathcal{C}_{T2T}$ indicates that there exists at least one tip-to-tip pattern at scan lines x_i and x_j in the clip. $\mathcal{C}_W$ is defined similarly for wire patterns. x_0 and y_0 define the origin of each clip that can be any value and do not affect pattern complexities. Certain constraint values correspond to the minimum critical dimensions when no defects are found in EUV simulation under a given process window [42]. Note that the system in Equation (10) can be efficiently solved with vast linear programming algorithms or numerical methods. Because, as discussed previously, all shapes will occupy the entire track in y direction, pitch and wire width are both covered by fixed track width and so is $\boldsymbol{\delta}_y$. We only need to consider constraints on $\boldsymbol{\delta}_x$ track by track. With the aid of squish representation, the problems of finding line-end-to-line-end patterns (for T2T constraints) and floating wires (for in-clip wire length constraints) become finding $\underbrace{100\cdots001}_{\text{continuous zeros}}$ and $\underbrace{011\cdots110}_{\text{continuous ones}}$ respectively. A solution of $\boldsymbol{\delta}_x$, $\boldsymbol{\delta}_y$ and the associated topology matrix $\boldsymbol{T}$ formulate a complete squish pattern representation. It should be also noted that a linear system like in Formula (10) tends to have multiple or infinite number of solutions. In our settings, only one solution is randomly selected for each generated topology. Actually, the diversity and the unique pattern count are calculated based on topologies and we introduce the linear system only to make the framework a complete pattern generation flow.

E. Overall Flow

We summarize the pattern generation flow as in Fig. 8, where key steps include squish pattern extraction, topology generation and pattern generation. In the topology generation phase, we force the TCAE to learn an identity mapping that can capture simple but important design rules. Such strategy also allows us to create a large fraction of new legal topologies by perturbing the latent vectors with either random noise or generative learning models. In the final pattern generation stage, we search critical dimensions that are defined in the design rules in all generated topologies and formulate corresponding linear systems to obtain legal $\boldsymbol{\delta}_x$s and $\boldsymbol{\delta}_y$s.

IV. Experimental Results

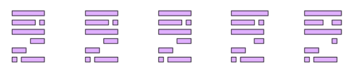
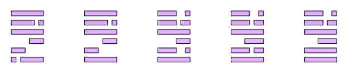
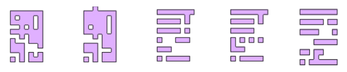
A. The Dataset and Configurations

We implement the pattern generation flow using `Python` and `Tensorflow` [43] library. The framework is tested on a platform with one Tesla P100 Graphic Card. We adopt five industry benchmark groups that contains metal-2 layout clips under $7nm$ EUV design node. The clip size is $192 \times 192nm^2$ and the corresponding squish topology size is zero-padded to 24×24 that will be the input size of the neural networks. The initial learning rate is set to 0.001 and decays by 0.7 every 2000 iterations. The maximum number of training steps is 10000 with a mini-batch size of 64. All neuron weights are initialized with Xavier [44] initializer and regularized with l_2 regularizer. The regularization coefficients for convolution layers are 0.001 and we chose 0.01 for densely connected layers. No data augmentation strategies are employed during training and the model at last training step is picked during inference stage. Note that we mark topologies with $c_x > 12$ and $c_y > 12$ as illegal such that the linear systems associated to legal topologies always *admit at least one* solution under the given window size, which ensures the quantity and quality of the generated pattern libraries. We adopt industrial solver when generating geometry information with Equation (10) for new topologies and only one solution is kept for each topology. Regarding the GAN component, the generator is initialized with Xavier without any regularization and the discriminator is l_2 regularized with coefficient of 0.01. The learning rate for GAN is set to be 0.001 and decayed by 0.05 every 10,000 iterations.

B. Understanding Features in TCAE

In the first experiment, we study the relationship between auto-learned features and human understandable layout space. Because the TCAE is trained to reconstruct input topologies as accurate as possible, feature vectors derived from the latent vector layer must attain all geometry informations that include wire tracks, line-end alignments, tip-to-tip distances, shape directions and so on. To show exactly how these auto-learned features affect the topology space, we conduct simple transformations on each individual entry of the latent vector and keep everything else unchanged. The transformed feature vectors are then fed into the reconstruction unit for topology reconstruction. We visualize part of the reconstructed patterns in Table. I, where each row corresponds to the transformation on certain nodes in the latent vector with everything else unchanged. In our case, a small perturbation is added up to a specific entry. We can easily observe that some features extend or pull back line-ends, some features create or destroy geometries and some features controls the directions of shapes. Unlike traditional design rules, auto-learned features control

TABLE I
Visualizing How Convolutional Features Are Reflected in Original Topology Space

Transformations	Reconstructed Topologies
Extend or pull back line-ends	
Create or destroy shapes	
Control shape directions	

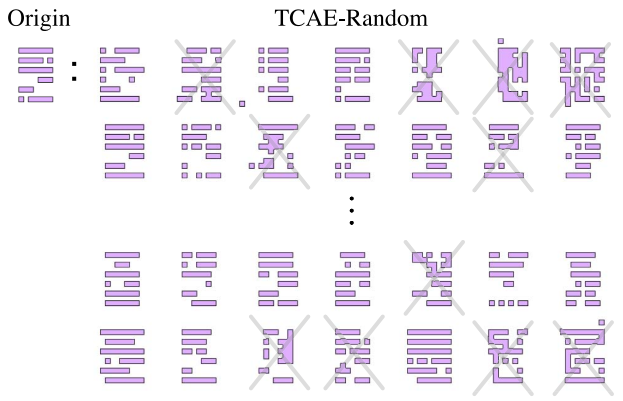


Fig. 9. Contribution of Gaussian perturbation on topology reconstruction. 1000 topologies (~400 legal) are created from one topology randomly picked from the existing pattern library.

layout patterns in a more global point of view that some features determine spacing, wire length and sometimes geometry direction as a whole.

Here we show how random perturbations on the latent vectors contribute to topology generation. We randomly take one topology from the training set and obtain its feature vector through the trained encoder network. 1000 noise vectors sampled from Gaussian are added up to the feature vector before it is fed into the decoder network for pattern reconstruction. We visualize the generated topologies in Fig. 9. We can observe that perturbations create significantly amount of new topologies where a large fraction of them are consistent with EUV pattern rules (e.g., no bow-tie shapes and unique direction single track polygons). Such results also show that deconvolution layers have the ability to learn simple design rules during training. On the contrary, no valid topology will be generated if the noise are directly applied on pattern space.

C. Evaluation of TCAE

We have shown the manipulation in the latent vector space can generate new topologies. In this experiment, we will make use of the flow above to augment the pattern space. Pattern library statistics are listed in Table II, where we randomly pick one pattern group to show the advantage of TCAE-Random. Column "Method" denotes the approach used to

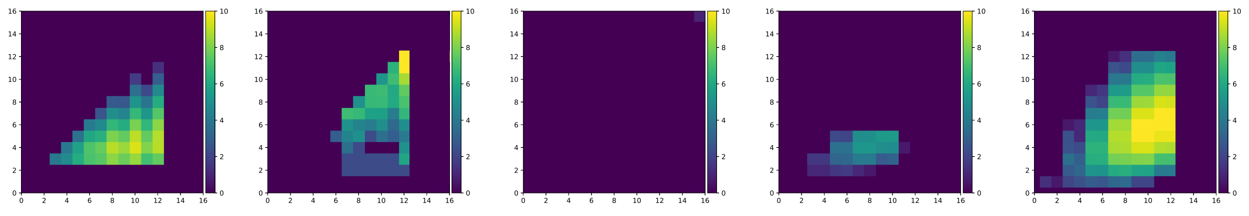

(a) Existing Design (b) Industry Tool (c) DCGAN (d) TCAE-Combine (e) TCAE-Random

Fig. 10. Visualization of the distribution of layout libraries: (a) Existing layout pattern dataset. (b) Industrial layout generator; (c) Patterns generated by DCGAN; (d) Patterns generated by TCAE-Combine; (e) Patterns generated by TCAE-Random.

TABLE II
STATISTICS OF GENERATED PATTERNS

Method	Pattern #	Pattern Diversity (H)
Existing Design	-	3.101
Industry Tool	55408	1.642
DCGAN	$\leq$10	0
VAE	4797	3.161
TCAE-Combine	1738	2.665
TCAE-Random	286898	3.337
TCAE-Random-r[1]	259746	3.486

1 Reduced generation count to match total chip area with "Industry Tool".

generate layout patterns, column "Pattern #" denotes the number of DRC clean patterns that are different from others, and column "Pattern Diversity" corresponds to the Shannon Entropy of each pattern library in terms of pattern complexity. Row "TCAE-Random" corresponds to the details of 1M patterns generated by perturbing the features of 1000 patterns in existing design with Gaussian noise. Row "TCAE-Combine" represents patterns generated from 1M different combinations of 10 test layout clip features. "Industry Tool" shows the cataloged results of a test layout generated from a state-of-the-art industry layout generator that are used internally. The test layout has similar total chip area (10000 μm^2) compared to "TCAE-Random-r" (9978 μm^2) and is smaller than "TCAE-Random" (14807 μm^2). We also implement a DCGAN [36] that has similar number of trainable parameters as the TCAE designed in this paper. Row "VAE" corresponds to a variational auto-encoder implementation to demonstrate our discussion in Section III-B4. The VAE has the same architecture as TCAE except the bottleneck layer that is replaced by VAE mean/variance vectors. The detailed configurations are following one state-of-the-art VAE design in [41]. 1M patterns are generated by feeding random latent vectors in the trained generator networks.

"Existing Design" lists the statistics of a pattern library extracted from an industry layout. Perturbation with Gaussian exhibits greatest pattern generation power with around 30% generated patterns are unique and DRC clean, thanks to the sensitive-aware latent node perturbation. Combination of patterns in feature space shows much less unique pattern count because the generation procedure are restricted by existing pattern space. Intuitively, TCAE-Combine tends to output most DRC-clean patterns. However, statistics only record <2000 unique DRC-clean patterns as linear combination of existing patterns are not likely to create new topologies. Combining more patterns will not affect the result much which will significantly reduce the count of DRC clean patterns if any two candidate patterns contain unaligned wires. Most GAN generated patterns fail with bow-tie or 2D wires even the training procedure has reached the equilibrium point because it is very hard to learn layout track information with randomly generated latent vectors. VAE, although, behaves better than a vanilla GAN, we can still see that the generated valid pattern count and diversity are very limited compared to TCAE-Random.

Fig. 10 compares the distributions of generated patterns and existing layout data set with similar pattern count, where x-axis and y-axis denote pattern complexity in each direction and the heatmap value is the total count (`log`-scale) of the pattern with that complexity. We employ *Pattern Diversity* to measure the pattern library distribution. We observe that Random perturbation can efficiently expand the weakly distributed pattern library (large fraction of patterns falls in certain complexities) with $H = 3.337$ while the industrial layout generator are still weakly distributed with $H = 1.642$ compared to existing designs.

D. Evaluation of GAN-Guided TCAE

We will evaluate the performance of G-TCAE in two aspects: (1) massive pattern generation and (2) context-specific pattern generation.

1) Massive Pattern Generation: In the massive pattern generation experiments, we train the TCAE model with the same settings as previous experiments. We dump out perturbation vectors that are used to create new valid patterns in TCAE-test phase that will serve as training source of the GAN. 1M of patterns are then created with GAN-generated perturbations. The results are listed in Table III, where columns "Pattern Diversity (H)" are calculated dataset pattern diversity in terms of Equation (1), columns "Pattern #" are unique DRC-clean pattern count amount 1M generated patterns, column "Benchmarks" lists five benchmark groups `directprint1`-`directprint5`, column "Training Set" corresponds to the statistics of the data used for training G-TCAE, column "TCAE" lists the performance of the original TCAE framework on five benchmarks, column "G-TCAE" corresponds to the performance of the proposed G-TCAE framework, and column "V-TCAE" corresponds to a case study result when we replace the GAN in G-TCAE with another prevailing generative model VAE.

TABLE III
RESULT COMPARISON BETWEEN TCAE-RANDOM [45] AND G-TCAE ON MASSIVE PATTERN GENERATION TASK

Benchmarks	Training Set	TCAE-Random [45]		G-TCAE		V-TCAE	
	Pattern Diversity (H)	Pattern #	Pattern Diversity (H)	Pattern #	Pattern Diversity (H)	Pattern #	Pattern Diversity (H)
`directprint1`	2.28	472259	3.48	491901	3.41	**563907**	**3.53**
`directprint2`	2.29	452892	3.67	**494784**	**3.72**	476765	3.52
`directprint3`	3.16	300019	3.83	313679	**3.84**	**421765**	3.68
`directprint4`	3.18	346233	**3.69**	382069	**3.69**	**402280**	3.58
`directprint5`	3.62	408738	3.83	**413060**	3.84	256803	**3.87**
Average	2.91	396028	**3.70**	419098	**3.70**	**424304**	3.64
Ratio	0.786	1.000	**1.000**	1.058	**1.000**	**1.071**	0.982

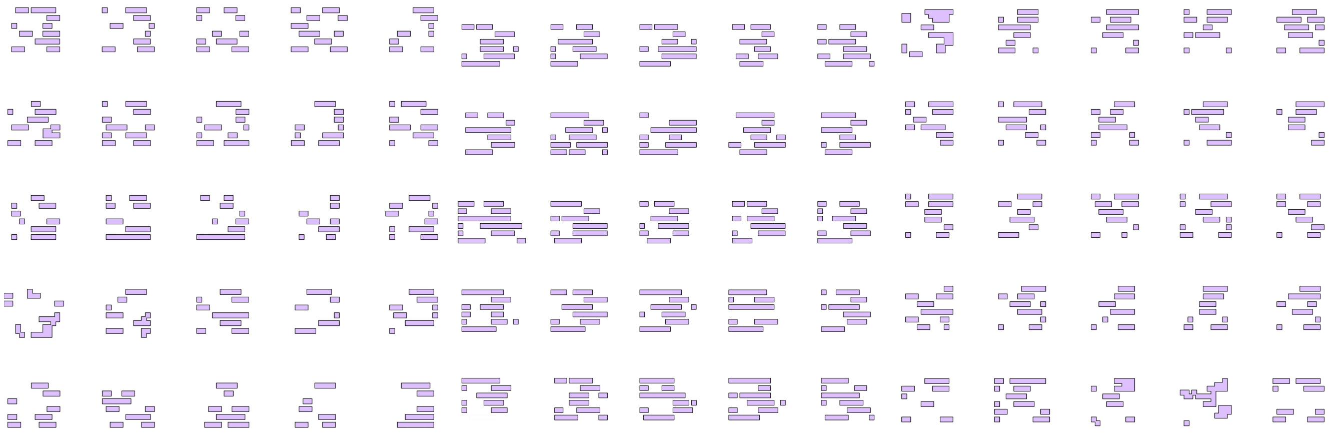

(a) Low complexity ($\bar{c_x} = 9.3, \bar{c_y} = 11.0$) (b) Medium complexity ($\bar{c_x} = 10.3, \bar{c_y} = 11.5$) (c) High complexity ($\bar{c_x} = 11.0, \bar{c_y} = 11.6$)

Fig. 11. Context specific pattern generation for different complexities.

From the table we can observe that both "TCAE" and "G-TCAE" successfully enlarges the layout pattern space by increasing the pattern diversity from 2.91 to 3.70, which demonstrates the validness of TCAE-family. By comparing the results of G-TCAE and TCAE, we show that G-TCAE offers much more unique DRC-clean patterns, with the help of GAN that transforms random perturbation vectors into design-rule-preserving vectors. On average, G-TCAE offers $\sim$ 5.8% more DRC-clean patterns than TCAE. We also observe that G-TCAE exhibits similar pattern diversity compared to TCAE, which can be explained by the fact that the GAN component is trained with perturbation vectors that are used for TCAE pattern generation. Similar pattern diversity will hence be expected when the GAN is trained well to an optimal state ($p_x = p_{\text{data}}$). As we have discussed in Section III-C, other generative models will also complete perturbation generation tasks in G-TCAE framework. Here, we conduct related experiments by replacing GAN in G-TCAE with a standard VAE (V-TCAE) and we can observe similar behavior compared to G-TCAE (424304/3.64 of V-TCAE vs. 419098/3.70 of G-TCAE).

2) Context-Specific Pattern Generation: Regarding the context-specific pattern generation, we choose `directprint1` as an example for performance evaluation. The first step is still the training of TCAE, after which latent vectors are divided into groups according to their pattern complexities. The GAN is then trained with these different latent vector groups and yields new vectors for certain complexity generation. We visualize our results in Fig. 11, that contains mixes of low, medium and high pattern complexities in x and y directions. Here, we also calculate the average pattern complexity to quantitatively evaluate such task. The generated three groups have the average c_x of 9.3, 10.3 and 11, respectively. Note that the average complexities of three groups in y direction are all around 11 $\sim$ 12 because patterns in the training set are mostly with $c_y = 11$ or 12.

V. CONCLUSION AND FUTURE WORKS

In this paper, we address the pattern generation problem in DFM flows/research for advanced technology nodes. We propose a transforming convolutional auto-encoder framework that can capture layout design rule characteristics. We show individual element in latent vector instantiation contributes to form the pattern space locally or globally, which inspires a pattern generation flow by perturbing the latent vector space. For the perspective of massive diverse DRC-clean pattern generation and context specific pattern generation, we propose a GAN-guided TCAE framework that further enhances the performance and functionality of TCAE-family. The experimental results show that our framework outperforms a state-of-the-art industrial layout generation tool in terms of pattern library diversity, which is promising to facilitate early technology node development and the back-end and sign-off flows. Future work includes expansion of TCAE family to complicated 2D designs, hotspot detection library optimization and large scale test design generation.

References

[1] G. R. Reddy, M.-M. Bidmeshki, and Y. Makris, "Viper: A versatile and intuitive pattern generator for early design space exploration," in *Proc. IEEE Int. Test Conf. (ITC)*, 2019, pp. 1–7.

[2] C. Tabery *et al.*, "In-design and signoff lithography physical analysis for 7/5nm," in *Proc. SPIE Adv. Lithogr.*, vol. 10147, 2017, Art. no. 1014705.

[3] J. Xu *et al.*, "Design layout analysis and DFM optimization using topological patterns," in *Proc. SPIE*, vol. 9427, 201, Art. no. 94270Q.

[4] A. Hamouda *et al.*, "Enhanced opc recipe coverage and early hotspot detection through automated layout generation and analysis," in *Proc. Opt. Microlithogr. XXX*, vol. 10147, 2017, Art. no. 101470R.

[5] Y. Lin *et al.*, "Data efficient lithography modeling with transfer learning and active data selection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 10, pp. 1900–1913, Oct. 2019.

[6] P. Kareem, Y. Kwon, and Y. Shin, "Layout pattern synthesis for lithography optimizations," *IEEE TSM*, vol. 33, no. 2, pp. 283–290, May 2020.

[7] Y. Du, H. Zhang, M. D. F. Wong, and K.-Y. Chao, "Hybrid lithography optimization with e-beam and immersion processes for 16nm 1D gridded design," in *Proc. ASPDAC*, 2012, pp. 707–712.

[8] H. Yang, L. Luo, J. Su, C. Lin, and B. Yu, "Imbalance aware lithography hotspot detection: A deep learning approach," *JM3*, vol. 16, no. 3, 2017, Art. no. 033504.

[9] J.-W. Jeon *et al.*, "Early stage hot spot analysis through standard cell base random pattern generation," in *Proc. Design-Process-Technol. Co-Optim. Manufacturability XI*, vol. 10148, 2017, Art. no. 101480S.

[10] B. Jiang, H. Zhang, J. Yang, and E. F. Young, "A fast machine learning-based mask printability predictor for OPC acceleration," in *Proc. ASPDAC*, 2019, pp. 412–419.

[11] M. Zhang *et al.*, "A weak pattern random creation and scoring method for lithography process tuning," in *Proc. Design-Process-Technol. Co-Optim. Manufacturability XII*, vol. 10588, 2018, Art. no. 105880U.

[12] H. Yang, J. Su, Y. Zou, Y. Ma, B. Yu, and E. F. Y. Young, "Layout hotspot detection with feature tensor generation and deep biased learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 6, pp. 1175–1187, Jun. 2019.

[13] M. Shin and J.-H. Lee, "Accurate lithography hotspot detection using deep convolutional neural networks," *JM3*, vol. 15, no. 4, 2016, Art. no. 043507.

[14] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Detecting multilayer layout hotspots with adaptive squish patterns," in *Proc. ASPDAC*, 2019, pp. 299–304.

[15] H. Geng, H. Yang, Y. Ma, J. Mitra, and B. Yu, "SRAF insertion via supervised dictionary learning," in *Proc. ASPDAC*, 2019, pp. 406–411.

[16] H. Yang, Y. Lin, B. Yu, and E. F. Y. Young, "Lithography hotspot detection: From shallow to deep learning," in *Proc. SOCC*, 2017, pp. 233–238.

[17] H. Yang, S. Li, C. Tabery, B. Lin, and B. Yu, "Bridging the gap between layout pattern sampling and hotspot detection via batch active learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 7, pp. 1464–1475, Jul. 2021.

[18] G. R. Reddy, K. Madkour, and Y. Makris, "Machine learning-based hotspot detection: Fallacies, pitfalls and marching orders," in *Proc. ICCAD*, 2019, pp. 1–8.

[19] L. Zhuang *et al.*, "A novel methodology of process weak-point identification to accelerate process development and yield ramp-up," in *Proc. ICSICT*, 2016, pp. 852–855.

[20] J. P. Cain, M. Fakhry, P. Pathak, J. Sweis, F. E. Gennari, and Y.-C. Lai, "Pattern-based analytics to estimate and track yield risk of designs down to 7nm," in *Proc. SPIE Adv. Lithogr.*, vol. 10148, 2017, Art. no. 1014805.

[21] H. Yang, W. Chen, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Automatic layout generation with applications in machine learning engine evaluation," 2019, *arXiv:1912.05796*.

[22] P. Kareem and Y. Shin, "Synthesis of lithography test patterns using machine learning model," *IEEE Trans. Semicond. Manuf.*, vol. 34, no. 1, pp. 49–57, Feb. 2021.

[23] M. Shafee *et al.*, "Approaches for full coverage physical design space exploration and analysis by synthetic layout generation," in *Proc. Design-Process-Technol. Co-Optim. Manufacturability XIV*, vol. 11328, 2020, Art. no. 1132808.

[24] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. NIPS*, 2014, pp. 2672–2680.

[25] M.-Y. Liu and O. Tuzel, "Coupled generative adversarial networks," in *Proc. NIPS*, 2016, pp. 469–477.

[26] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. ICCV*, 2017, pp. 2242–2251.

[27] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014, *arXiv:1411.1784*.

[28] W. Ye, M. B. Alawieh, Y. Lin, and D. Z. Pan, "LithoGAN: End-to-end lithography modeling with generative adversarial networks," in *Proc. DAC*, 2019, pp. 1–6.

[29] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *Proc. DAC*, 2018, p. 131.

[30] H. Geng, H. Yang, B. Yu, X. Li, and X. Zeng, "Sparse VLSI layout feature extraction: A dictionary learning approach," in *Proc. ISVLSI*, 2018, pp. 488–493.

[31] M. B. Alawieh, Y. Lin, Z. Zhang, M. Li, Q. Huang, and D. Z. Pan, "GAN-SRAF: Sub-resolution assist feature generation using conditional generative adversarial networks," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–6.

[32] J. Liu, Y. Ding, J. Yang, U. Schlichtmann, and Y. Shi, "Generative adversarial network based scalable on-chip noise sensor placement," in *Proc. SOCC*, 2017, pp. 239–242.

[33] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. ICML*, 2017, pp. 214–223.

[34] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," in *Proc. ICLR*, 2016, pp. 1–17.

[35] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein GANs," in *Proc. NIPS*, 2017, pp. 5767–5777.

[36] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *Proc. ICLR*, 2016, pp. 1–16.

[37] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *Proc. ICANN*, 2011, pp. 44–51.

[38] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 5, no. 1, pp. 3–55, 2001.

[39] F. E. Gennari and Y.-C. Lai, "Topology design using squish patterns," U.S. Patent 8 832 621, Sep. 9, 2014.

[40] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2018, *arXiv:1603.07285*.

[41] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013, *arXiv:1312.6114*.

[42] P. Gupta, "What is process window?" *ACM SIGDA Newslett.*, vol. 40, no. 8, p. 1, 2010.

[43] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, 2016, pp. 265–283.

[44] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. AISTATS*, vol. 9, 2010, pp. 249–256.

[45] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "DeePattern: Layout pattern generation with transforming convolutional auto-encoder," in *Proc. DAC*, 2019, pp. 1–6.