LayouTransformer: Generating Layout Patterns with Transformer via Sequential Pattern Modeling

Liangjian Wen*
Huawei Noah's Ark Lab
wenliangjian1@huawei.com

Yi Zhu*
Huawei Noah's Ark Lab
zhuyi36@huawei.com

Lei Ye
Huawei Noah's Ark Lab
yelei13@huawei.com

Guojin Chen
CUHK
glchen@cse.cuhk.edu.hk

Bei Yu
CUHK
byu@cse.cuhk.edu.hk

Jianzhuang Liu
Huawei Noah's Ark Lab
liu.jianzhuang@huawei.com

Chunjing Xu
Huawei Noah's Ark Lab
xuchunjing@huawei.com

ABSTRACT

Generating legal and diverse layout patterns to establish large pattern libraries is fundamental for many lithography design applications. Existing pattern generation models typically regard the pattern generation problem as image generation of layout maps and learn to model the patterns via capturing pixel-level coherence, which is insufficient to achieve polygon-level modeling, e.g., shape and layout of patterns, thus leading to poor generation quality. In this paper, we regard the pattern generation problem as an unsupervised sequence generation problem, in order to learn the pattern design rules by explicitly modeling the shapes of polygons and the layouts among polygons. Specifically, we first propose a sequential pattern representation scheme that fully describes the geometric information of polygons by encoding the 2D layout patterns as sequences of tokens, i.e., vertexes and edges. Then we train a sequential generative model to capture the long-term dependency among tokens and thus learn the design rules from training examples. To generate a new pattern in sequence, each token is generated conditioned on the previously generated tokens that are from the same polygon or different polygons in the same layout map. Our framework, termed LayouTransformer, is based on the Transformer architecture due to its remarkable ability in sequence modeling. Comprehensive experiments show that our LayouTransformer not only generates a large amount of legal patterns but also maintains high generation diversity, demonstrating its superiority over existing pattern generative models.

1 INTRODUCTION

Diverse layout patterns of Very-Large-Scale Integration (VLSI) are of crucial significance for a variety of lithography design applications, including OPC recipes [1–4], source mask optimization, layout hotspot detection [5–8], and lithography simulation [9–11].

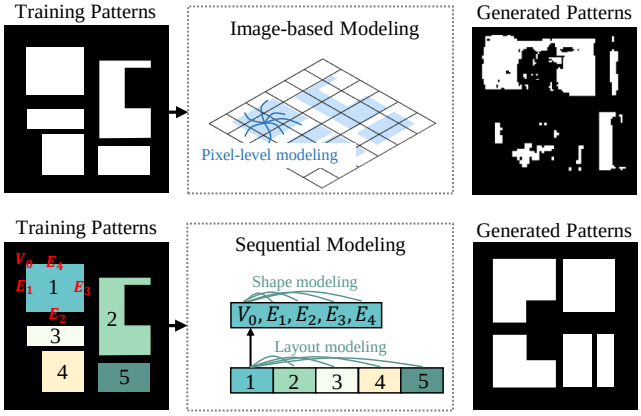


Figure 1: Illustration of existing image-based modeling and our proposed sequential modeling for pattern generation.

In recent years, machine learning is well adopted in various lithography design applications, giving rise to the requirement for large-scale and diverse layout patterns. Building such pattern libraries is extremely time-consuming due to the long logic-to-chip design cycle. To overcome this, researchers begin to pay more attention on synthesizing diverse layout patterns of VLSI.

Early works [12–14] propose simple deterministic methods to synthesize new patterns. For example, some predefined unit patterns are manually selected to be combined into a new pattern. Another simpler method is to rotate or flip exiting patterns to obtain new patterns. Though effective in some extent, they only achieve minor augmentation for the original pattern library, and both the quantity and diversity of the synthesized patterns are limited.

In order to break this limitation, researchers exploit powerful machine learning methods to model the design process of patterns through deep neural networks. They regard the pattern generation problem as image generation of layout maps, each with multiple polygons. Nevertheless, image-based modeling for layout pattern generation may suffer from two drawbacks. First, the pixel-level modeling of pattern geometric information is insufficient to capture the shapes and layouts of patterns, leading to poor legality of the generated patterns, as shown in Figure 1 (upper). To alleviate this, time-consuming post-processing is needed, which smoothes the noisy patterns based on a Conditional Generative Adversarial Network (CGAN) [15] to reduce the risk of violating DRC rules. Second, the diversity of patterns generated by existing image-based generative models is limited. Generative Adversarial Networks (GANs)

*Equal contribution

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10…$15.00
https://doi.org/10.1145/3508352.3549350

[16] suffer from model collapse, which significantly reduces the diversity of generated patterns. The Convolutional Auto-Encoders (CAEs) [17] use the squish pattern representation [18] (splitting a layout map into a topology matrix and two geometry vectors), and conducts perturbation on the topology matrices of training patterns to generate new diverse patterns. The diversity is still limited, since the new patterns must have the same geometric vectors as the existing examples.

In this work, we tackle the pattern generation problem from a brand new perspective of sequential modeling, which is inspired by the remarkable progress in natural language processing, e.g., GPT [19], BERT [20], and XLNet [21]. These Transformer-based models show great potential in modeling the complicated grammar rules of language sentences. So, what would happen if we regard the pattern design rules as a set of grammar rules and build a pattern generative model taking the advantages of successful language generation models? The key to this idea is to design a highly structured sequential representation that explicitly describes the shapes and spatial layouts of the patterns, with the design rules learnt during training. To this end, we propose a novel pattern sequential representation that converts the 2D layout patterns into 1D sequences. As shown in Figure 1 (lower), each polygon is represented as a sequence containing the coordinates of the starting point V_0 (upper-left corner) and the edges $\{E_1, E_2, ...\}$ for walking through the polygon boundary from V_0 in a counterclockwise direction. The polygons are sorted in ascending order in the sequence of the layout map according to the starting point. Such sequential pattern representation is lossless and has high coding efficiency. Our sequential modeling explicitly encodes the shapes and spatial layouts of patterns as 1D sequences, which are more friendly for generative modeling with a Transformer.

Note that the pattern design rules and grammar rules have similar properties from the sequential modeling perspective. First, language generation requires the model to capture the long-term dependencies among the words in a sentence, and pattern generation models need to capture the dependencies among each edge of the patterns. Second, each token is generated conditioned on the previously generated ones to maintain continuity of language meaning or polygon shape, resulting in new legal sequences. Third, when the stop token is predicted, a sequence is completed, and thus a whole sentence or a closed polygon is obtained.

We develop our sequential pattern generation framework, termed LayouTransformer, based on the Transformer [22] architecture, which is shown to be excellent at sequence modeling and is widely-used in generating sequences of arbitrary lengths. During training, LayouTransformer models the dependencies among tokens of pattern sequences and learns the pattern design rules from traning patterns. After that, LayouTransformer generates a new pattern by sampling each token from the learnt probability distribution over the vocabulary (including all possible token values) conditioned on the previously generated tokens.

Our main contributions are summarized as follows:

- We tackle the challenging pattern generation problem from a brand new perspective of sequential modeling.
- We propose sequential patterns, a novel, lossless, and highly structured representation strategy, to explicitly encode the shape and spatial information of patterns.

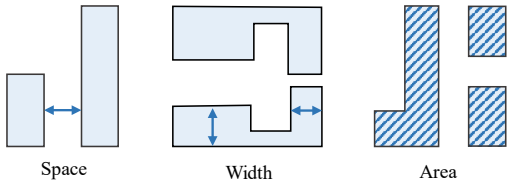


Figure 2: Layout geometry concepts for DRC rules.

- We develop a Transformer-based sequential generative model to learn the pattern design rules and generate new diverse layouts.
- Comprehensive experiments are conducted to show the superior performance of our method on pattern legality, diversity, and validity.

2 RELATED WORK

Synthesizing diverse layout patterns of VLSI has obtained increasing attention for lithography design applications. [23] applies Generative Adversarial Networks (GANs) [16] to generate discrete cosine transform (DCT) signals, which can be converted to layouts via the inverse DCT. However, the polygons of synthesized patterns are irregular, and the complex post-process is required. Moreover, GANs are notoriously difficult to train in practice due to the instability of gradient-based mini-max optimization. Besides, GANs also suffer from model collapse, which seriously limits the diversity of synthesized patterns.

CAE is an alternative for layout pattern generation. [17] utilizes convolutional auto-encoder to generate new patterns by adding perturbation to the latent representations of existing ones. This work aims to generate uni-direction patterns. To establish diverse, large, and DRC-clean pattern libraries for complex 2D layouts, [24] proposes a two-stage training framework for pattern topology generation and legalization. This work firstly trains a variational convolutional auto-encoder to generate new patterns. Then a CGAN [15] is designed to make blurry generated patterns smooth to reduces DRC violation risks. [25] attempts to employ an Adversarial Auto-Encoder (AAE) [26] to generate IPS (image parameter space) values of layout patterns, which can be mapped to layout patterns. However, the polygons of synthesized patterns are irregular, which need complex post-processing.

3 PATTERN GENERATION PROBLEM

In this section, we describe the fundamental concepts and evaluation metrics for pattern generation. The target of pattern generation is to synthesize diverse and realistic layout patterns based on a set of real IC layout patterns.

Layout Design Rules. The generated patterns are required to follow the layout design rules of actual IC layouts. As depicted in Figure 2, "Space" measures the distance between two adjacent polygons, "Width" measures the size of a shape in one direction, and "Area" represents the area of a polygon. If a layout pattern is legal, these geometric measurements need to satisfy some given critical values in the layout design rules.

Pattern Diversity. The diversity of layout patterns is a crucial evaluation metric for pattern generation. As defined in [17], the complexity of a layout pattern is represented as (c_x, c_y), where c_x

and c_y are the numbers of scan lines subtracted by one along the x-axis and y-axis, respectively. Hence, following [17], the pattern diversity is defined as the Shannon entropy [27] of the distribution of the pattern complexities as follows:

Definition 1 (Pattern Diversity). *Pattern diversity, denoted by H, is the Shannon entropy of the pattern complexity sampled from the pattern library,*

$$H = -\sum_i \sum_j P\left(c_{xi}, c_{yj}\right) \log P\left(c_{xi}, c_{yj}\right), \tag{1}$$

where $P\left(c_{xi}, c_{yj}\right)$ denotes the probability of a pattern with complexity (c_{xi}, c_{yj}) sampled from the library.

According to this definition, the larger the entropy of the pattern complexity in the pattern library, the more diverse the patterns.

Pattern Validity. Synthesizing realistic layout patterns is another essential target of pattern generation. This implies the generated patterns should closely resemble the real patterns. To evaluate how realistic the generated patterns are, [24] defines pattern validity as follows:

Definition 2 (Pattern Validity). *Pattern validity is the ratio of realistic patterns to total patterns.*

Section 6.3 gives a scheme to measure if a generated pattern is realistic. Based on the above evaluation metrics for pattern generation, the fundamental framework of pattern generation can be formulated as follows:

Definition 3 (Pattern Generation). *Given a set of real layout patterns and design rules, pattern generation aims to establish a legal pattern library such that the pattern diversity and the pattern validity of the layout patterns in the library are maximized.*

4 OVERVIEW OF OUR FRAMEWORK

Learning to generate diverse and legal layout patterns in 2D maps is challenging since it is difficult to capture and follow a lot of design rules about the geometric shapes and spatial layouts of the patterns. In this work, to the best of our knowledge, we for the first time regarding the 2D pattern generation problem as a 1D sequence generation problem. To achieve this, we design a novel sequential pattern generation framework that can well capture the pattern design rules and generate a large variety of new layout patterns.

As shown in Figure 4, given a set of layout examples, we first extract their sequential pattern representations, where each pattern is serialized as tokens of the starting point and the directions and offsets of edges. Then the extracted sequences are encoded and fed into a sequential generative model, which is built upon the Transformer architecture, a successful model in generating well-structured and grammar-correct sentences, in an auto-regressive manner. Training with a dataset of serialized patterns, our auto-regressive generative model can generate new layout patterns which are legal and diverse.

5 SEQUENTIAL PATTERN GENERATION

Transformers have shown great potential in sequence modeling due to its remarkable ability in capturing long-term dependencies of sequences. As a result, Transformers become a dominant architecture for natural language processing where the model is required to capture long sequence features, understand the structure of language, maintains common grammar rules, and meanwhile enable parallel training. Inspired by the unprecedented success of Transformer-based language generation models, e.g., GPT [19], BERT [20], and XLNet [21], we take advantage of the sequence modeling ability of the Transformer architecture and design a novel sequence generation framework for generating layout patterns.

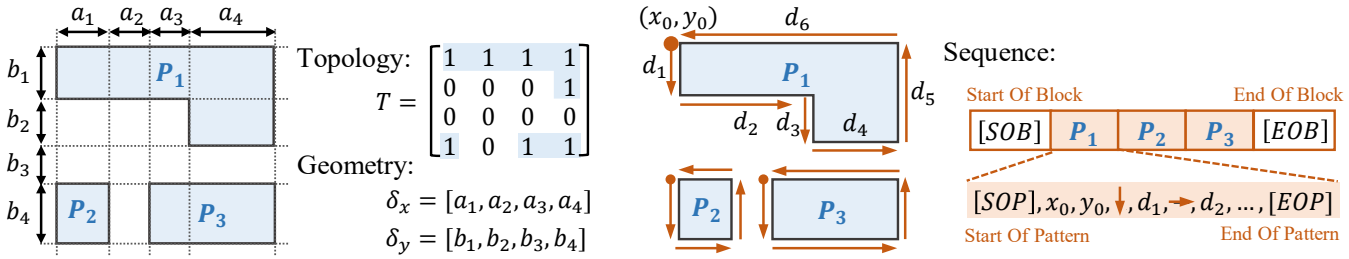
To achieve this, we first propose sequential pattern representation that converts the 2D layout patterns into 1D sequences, in Section 5.1. Then, we formulate the pattern generation problem from the perspective of sequential modeling (Section 5.2), and design a sequential pattern generation model based on the Transformer architecture (Section 5.3). The training and generating procedures are described in Section 5.4 and Section 5.5, respectively.

5.1 Sequential Pattern Representation

In this subsection, we first give a brief introduction of the recently proposed squish patterns [18] which is a 2D pattern representation for generative models. After that, we present our sequential pattern representation that converts the 2D patterns into 1D sequences.

Squish pattern representation compresses the layout images into a pattern topology matrix $\mathbf{T}$ and two geometric vectors $\boldsymbol{\delta}_x$ and $\boldsymbol{\delta}_y$. As shown in Figure 3(a), the topology $\mathbf{T}$ is a binarized matrix where 1 indicates the existence of a polygon and 0 otherwise. The vectors $\boldsymbol{\delta}_x$ and $\boldsymbol{\delta}_y$ describe the widths and heights of the grids in x and y axises, respectively. Such compositional representation requires the layout generation models to generate the topology matrix and the geometric vectors jointly and meanwhile to maintain the pattern legalization. This is quite challenging for modern generative models. Existing layout generation learning models [17, 24] learn to generate layout typologies with the geometric vectors fixed, which limits the diversity and flexibility of pattern generation.

In contrast, our proposed sequential patterns represent the layout maps in a more compact and flexible way. As shown in Figure 3(b), each polygon is represented as a sequence containing the coordinates of the starting point, and the directions and offsets for walking through all the edges in turn from the starting point in a counterclockwise direction. Specifically, each polygon is identified with a pair $[SOP]$ and $[EOP]$ tokens indicating the start and the end of the pattern sequence, respectively. The starting point (x_0, y_0) is the upper-left corner of the polygon. We define four directional tokens to represent the directions of "up", "down", "left" and "right". Each of the directional tokens is followed by an offset value indicating the length of the edge. The coordinates of the starting point and the offsets are discretized into the integers in $[1, N]$, with 1 being the minimum distance unit and N being the maximum length of the layout patterns. Multiple polygons are contained within a pair of tokens $[SOB]$ and $[EOB]$ denoting the head and the tail of a layout block. In the sequential representation of the layout block, the polygons are sorted in ascending order according to the y-coordinate of their upper-left corners. If two polygons have the same y-coordinate of their upper-left corners, the one with smaller x_0 comes first. In light of language, we define the vocabulary as a set of all possible values of the tokens in the pattern sequences. Hence, the size of vocabulary is equal to $N + 8$ where 8 is the number of



(a) Squish pattern representation (b) Sequential pattern representation

Figure 3: Overview of the squish pattern representation and our proposed sequential pattern representation.

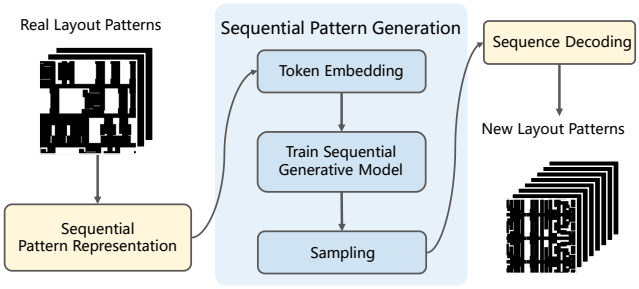


Figure 4: Overview of our LayouTransformer framework.

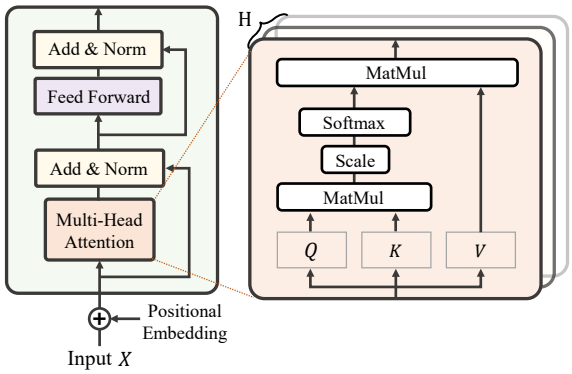


Figure 5: Illustration of the Transformer block.

the tokens including $[SOP]$, $[EOP]$, $[SOB]$, $[EOB]$, "up", "down", "left" and "right".

5.2 Sequential Modeling of Pattern Generation

Our pattern sequence representation is lossless and preserves the full geometric information of each polygon and the spatial relationship among different polygons. The geometric information implicitly contains the design rules of layout patterns. Hence, learning how to generate different tokens according to training samples allows the model to implicitly learn the design rules.

Let $\boldsymbol{t} = \{t_1, ..., t_L\} = \{[SOB], [SOP], x_0, y_0, ..., [EOB]\}$ be the sequence of a layout map. The design rules of patterns can be learned by modelling the joint distribution $p(\boldsymbol{t})$ of the sequence $\boldsymbol{t} = \{t_1, ..., t_L\}$. We cast the estimation of the joint distribution $p(\boldsymbol{t})$ to the autoregressive sequence generation. Based on the chain rule of probability, the join distribution $p(\boldsymbol{t})$ can be factored into:

$$p(\boldsymbol{t}) = p\left(t_1, \ldots, t_L\right) = \prod_{i=1}^{L} p\left(t_i \mid t_1, \ldots, t_{i-1}\right). \tag{2}$$

This decomposes the density estimation of $p(\boldsymbol{t})$ into next-token prediction (e.g., t_i) conditioned on previous tokens (e.g., $\{t_1, t_2, ..., t_{i-1}\}$). Transformer is adopted to learn the above product of conditional distributions due to its powerful ability of density estimation and modelling long-range complex dependencies. More specifically, the Transformer network θ takes in the conditional sequence of tokens $\{t_1, \ldots, t_{i-1}\}$ and outputs a categorical distribution over the possible values of the next token using the softmax function. Based on the principle of maximum likelihood, we maximize the log-probability of the true tokens $p\left(t_i \mid t_1, \ldots, t_{i-1}\right)$ to train the network θ.

The network θ can model the relationships among different t_i to learn the design rules of layout patterns. For example, regarding the currently generated sequence "$[SOP], x_0, y_0,$", our layout generation model predicts the next token as one of the directional tokens. This is similar to language generation where each word is predicted based on the previous words. Moreover, our generation model is required to correctly predict the ending token and meanwhile enable the closure of the polygon; i.e., following the generated directions and offsets, we can finally walk back to the starting point, and thus the trajectory can form a complete polygon with horizontal and vertical edges. Likewise, a grammar-correct sentence needs a correct ending signal and integrity of both the semantic meaning and linguistic structure.

5.3 Model Architecture

Transformer [28] has been proved excellent at capturing long-term dependencies among language sentences. The vanilla Transformer architecture consists of K blocks each with a multi-head attention layer, a fully connected feed forward network, and two normalization and residual add-up layers, as shown in Figure 5. We denote the Transformer as a function $\mathrm{TF}(\cdot)$: $\mathbb{R}^{N\times d} \rightarrow \mathbb{R}^{N\times d}$ which takes a sequence of token embeddings $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}, \boldsymbol{x}_i \in \mathbb{R}^d$ as input, and outputs a sequence of token embeddings $\hat{\boldsymbol{X}} = \{\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, ..., \hat{\boldsymbol{x}}_N\}, \hat{\boldsymbol{x}}_i \in \mathbb{R}^d$, where each $\hat{\boldsymbol{x}}_i$ encodes the dependencies between $\boldsymbol{x}_i$ and other input tokens. At first, the input token embeddings $\boldsymbol{X}$ are added with the positional embeddings to encode the position information of each token. Then, the multi-head attention module learns independent transformations $\boldsymbol{W}_h^K, \boldsymbol{W}_h^Q$ and $\boldsymbol{W}_h^V \in \mathbb{R}^{d\times d}$ at the h-th head and calculates the attention weights $\boldsymbol{A}_h$ between all input tokens. The token embeddings are updated with the context information from their attended tokens. Finally,

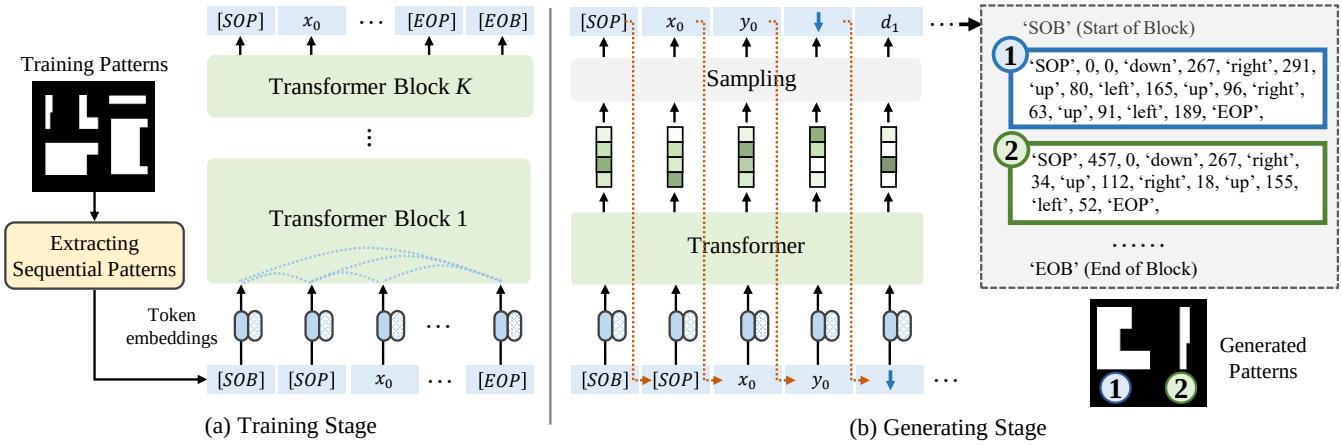


Figure 6: Training LayouTransformer and token generation.

the resulting token embeddings at all attention heads are concatenated and once again projected to input dimension via a learnable weight matrix $\boldsymbol{W}^p \in \mathbb{R}^{(H\times d)\times d}$. For each $\boldsymbol{x}_i \in \boldsymbol{X}$, we obtain the output embedding $\hat{\boldsymbol{x}}_i$ as:

$$\boldsymbol{A}_h(\boldsymbol{x}_i, \boldsymbol{x}_j) = \text{softmax}((\boldsymbol{W}_h^Q \boldsymbol{X})(\boldsymbol{W}_h^K \boldsymbol{X})^\top / \sqrt{d}), \tag{3}$$

$$\tilde{\boldsymbol{x}}_i = \text{concat}_{h=1}^{H} \left\{ \sum_{j=1}^{N} \boldsymbol{A}_h(\boldsymbol{x}_i, \boldsymbol{x}_j)(\boldsymbol{W}_h^V \boldsymbol{X}) \right\}, \tag{4}$$

$$\hat{\boldsymbol{x}}_i = \boldsymbol{W}^p \tilde{\boldsymbol{x}}_i. \tag{5}$$

Beyond the Transformer, our model introduces two extra mechanisms for the sequential pattern generation learning. The first one is masked self-attention, where each token is only allowed to attend to the tokens preceding it in the sequence, since the subsequent tokens are unknown currently during token generation. The second is segment-level recurrence, which is inspired by recurrent neural networks and first proposed in Transformer-XL [22]. In practice, if a sequence is very long, it is divided into several segments in a fixed length to reduce computation cost. This mechanism enables the learning of dependencies across segments without disrupting their coherence. By doing this, our model can well capture the long-term relationship among sequences of different polygons and even polygons from different layout maps.

5.4 Training LayouTransformer

Here we present the training process of our sequential pattern generation model, as shown in Figure 6(a). Given a layout map I, the patterns on it are first encoded as a sequence $\boldsymbol{t} = \{t_1, ..., t_L\} = \{[SOB], [SOP], x_0, y_0, ..., [EOB]\}$. The tokens in the sequence $\boldsymbol{t}$ are tokenized and fed into a learnable word embedding layer $f(\cdot)$, resulting in a set of embeddings $\boldsymbol{S} = \{\boldsymbol{s}_1, ..., \boldsymbol{s}_L\}, \boldsymbol{s}_i = f(t_i) \in \mathbb{R}^d$. The word embedding layer aims to learn an embedding for each token in the vocabulary V which contains all possible tokens. Also, we employ the positional embeddings to encode the relative positions of the tokens as in [22], obtaining a set of embeddings $\boldsymbol{E} = \{\boldsymbol{e}_1, ..., \boldsymbol{e}_L\}, \boldsymbol{e}_i \in \mathbb{R}^d$. Note that the positional embeddings are crucial to model the shapes and spatial layouts of patterns, since the relative position information of a token can provide the strong prior for predicting a token, e.g., the next token of $[SOB]$ must be $[SOP]$. The model learns to predict each token t_i based on the previous tokens $\{t_1, ..., t_{i-1}\}$. For example, as shown in Figure 6(a), the token $t_i = x_0$ in the sequence $\boldsymbol{t}$ is determined based on the given subsequence $\{[SOB], [SOP]\}$. First, each of the token embeddings in $\boldsymbol{S}$ and the positional embeddings in $\boldsymbol{E}$ are added together and fed into the Transformer model. Then, the model outputs $\{\boldsymbol{z}_1, ..., \boldsymbol{z}_{i-1}\}$ and uses the last hidden state $\boldsymbol{z}_{i-1}$ to predict token $\boldsymbol{t}_i$. This procedure is represented as:

$$\{\hat{\boldsymbol{z}}_1, ..., \hat{\boldsymbol{z}}_{i-1}\} = \{\boldsymbol{s}_1 + \boldsymbol{e}_1, ..., \boldsymbol{s}_{i-1} + \boldsymbol{e}_{i-1}\}, \tag{6}$$

$$\{\boldsymbol{z}_1, ..., \boldsymbol{z}_{i-1}\} = \text{TF}(\{\hat{\boldsymbol{z}}_1, ..., \hat{\boldsymbol{z}}_{i-1}\}), \tag{7}$$

where $\hat{\boldsymbol{z}}_i, \boldsymbol{z}_i \in \mathbb{R}^d$. We use the cross entropy loss between the conditional distribution $p(t_i \mid t_1, \ldots, t_{i-1})$ and the one-hot distribution of the target token t_i to maximize the prediction of t_i, thus optimizing the parameters of our model as:

$$p(t_i \mid t_1, \ldots, t_{i-1}) = \frac{\exp(\boldsymbol{z}_{i-1}^\top f(t_i))}{\sum_{t\in V} \exp(\boldsymbol{z}_{i-1}^\top f(t))}. \tag{8}$$

In the training process, the predictions of different tokens are independent, so they can be calculated in parallel, meaning that LayouTransformer can be trained efficiently.

5.5 Generating New Patterns

After training, our model can generate new patterns in an autoregressive manner, i.e., each token is generated based on the preceding generated ones, as shown in Figure 6(b). Given a starting prompt as input, e.g., "$[SOB], [SOP]$", the prompt is first tokenized and encoded via the learned word embedding layer, and then fed into the trained transformer blocks. Based on the output vectors, we can obtain the categorical distribution $p(t_3 \mid t_1 = [SOB], t_2 = [SOP])$ over the possible values of the next token using the softmax function. From the distribution, a token x_0 is sampled as the generated next token, which is then regarded as a input token for predicting the subsequent token. Then, "$[SOB], [SOP], x_0$" is fed back into the model for continuing generation. Repeat the generation process above until encountering the token $[EOB]$ which indicates the end of the generation of a layout map.

Here we employ the Nucleus Sampling [29] as our sampling strategy to enable generation diversity while maintaining legality. It selects a smallest possible set of top ranked tokens, such that the sum of their probabilities is greater than a threshold ϵ. The probability of the rest tokens are set to 0, and the probabilities of the tokens in the set are re-scaled to ensure the sum of the probabilities to be 1. From the rescaled distribution, we sample a token for generation. When the model is very certain on few tokens, the potential candidate set is small, e.g., generating a "[SOP]" regarding a "[SOB]". Otherwise, there will be many potential candidate tokens, e.g., generating "x_0" regarding a "[SOP]", sampling from which will generate diverse polygons in the block.

6 EXPERIMENTS

6.1 Experimental Setup

Dataset: We follow [23] to obtain the dataset of small layout pattern images with the size of $2048 \times 2048nm^2$ by splitting a $160 \times 400\mu m^2$ layout map from ICCAD contest 2014. 80% of the images are randomly selected as the training set while the others serve as the validation set for model training.

Network Configuration: In our sequential pattern generation framework, the vocabulary size is 2056, containing 2048 possible integers of coordinates and offsets, 4 directional tokens (up, down, left, right), and 4 tokens indicating the start or end of a polygon or a block. The dimensionality of the word and positional embeddings are set to 512. Our Transformer architecture has $K = 6$ Blocks, and each multi-head attention module has 8 attention heads. The lengths of memory and prediction of our Transformer model are both set to 512. The dimensionality of the hidden states is also 512, and the intermediate size of the feed-forward networks is 1024.

Training Details: Our sequential pattern generation is implemented with PyTorch [30]. The training of our model runs for 40K steps, about 42 epochs with batch size 22. We employ the Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate is set to $3e-4$, with a warm-up of 2000 steps and the cosine learning rate decay. The rates of the residual dropout and attention dropout are both set to 0.1. The total training procedure takes about four hours using 8 Nvidia Telsa V100 32GB GPUs.

Table 1: Comparison with recent learning-based methods.

Set/Method			Legal Patterns	
	Patterns	Diversity (H)	Patterns	Diversity (H)
Real patterns	13869	10.7767	–	–
CAE [17]	100000	4.5875	19	3.7871
VCAE [24]	100000	**10.9311**	2126	9.9775
CAE+LegalGAN [24]	100000	5.8465	3740	5.8142
VCAE+LegalGAN [24]	100000	9.8692	84510	9.8669
LayouTransformer (Ours)	100000	10.532	**89726**	**10.527**

6.2 Pattern Diversity and Legality

We generate 100000 patterns from the trained model and evaluate their diversity and legality. The legality is checked via a tool *Klayout* based on the design rules described in Section 3. After that, we obtain the DRC-clean legal patterns and we further evaluate their diversity.

We compare our LayouTransformer with several learning-based baselines. CAE [17] is a vanilla convolutional auto-encoder model. VCAE [24] is a variational convolutional auto-encoder model. The LegalGAN [24] model can legalize most of the illegal patterns. In Table 1, the VCAE baseline achieves larger diversity (10.9311) than our method (10.532) on the generated patterns. However, most of the VCAE generated patterns violate the DRC rules (only 2126 patterns are legal) and cause an invalid larger diversity value. Furthermore, we can see that the our method obtains the largest number of DRC-clean patterns (89726 out of 100000) and the best diversity performance (10.527) on the legal patterns, which demonstrates the superiority of our sequential pattern generation model. Note that we do not use LegalGAN to correct illegal patterns. Because our sequential pattern representation explicitly encodes the complete geometric information of each polygon and the spatial layout of multiple polygons, our trained LayouTransformer can well capture the design rules and generate DRC-clean patterns. The compared baselines typically use the squish pattern representations and learn a model to generate the topology map with the geometry fixed, which limits the diversity of pattern generation in some extent. In contrast, since LayouTransformer models the distribution of layout patterns both topologically and geometrically, sampling from it enables better diversity.

We show some randomly selected examples of the training and generated patterns in Figure 7 and Figure 8, respectively. From these examples, we can observe that: 1) different from VCAE, where the geometry vectors only come from the training patterns, our method can generate patterns with new geometries that are never seen in the training set; 2) the complexity of our generated polygons are consistent with the training samples, without an undesirable significant style gap between them; 3) when a layout map contains many complex patterns, they can be well arranged at legal locations regarding their surrounding patterns.

Table 2: Pattern validity comparison. The 10787 training patterns from the real patterns are used to train the detection model [23].

Set/Method	Legal Patterns	Pattern Validity		
		$T = 0.6$	$T = 0.7$	$T = 0.8$
Training patterns	10787	0.8998	0.9702	0.9904
CAE+LegalGAN [24]	3740	0.0003	0.0027	0.0167
VCAE+LegalGAN [24]	84510	0.5430	0.7840	0.9057
LayouTransformer (Ours)	**89726**	**0.8416**	**0.9438**	**0.9834**

6.3 Pattern Validity

In this subsection, we evaluate how realistic the patterns generated by LayouTransformer are. We adopt the pattern style detection model proposed by [24] to verify the validity of the patterns.

The key idea of the model is that realistic patterns with a particular layout style are viewed as normal; otherwise they are anomalous. Hence, pattern style detection is regarded as anomaly detection of generated layouts. Specifically, a CGAN [15] is trained to learn the distribution of normal samples. Since the encoder is trained on normal samples, it fails to learn anomalous features. The trained generator is unable to reconstruct an anomalous pattern if it is encoded. Hence, we can detect if a given pattern is realistic or not by the difference of features between the pattern and its reconstructed

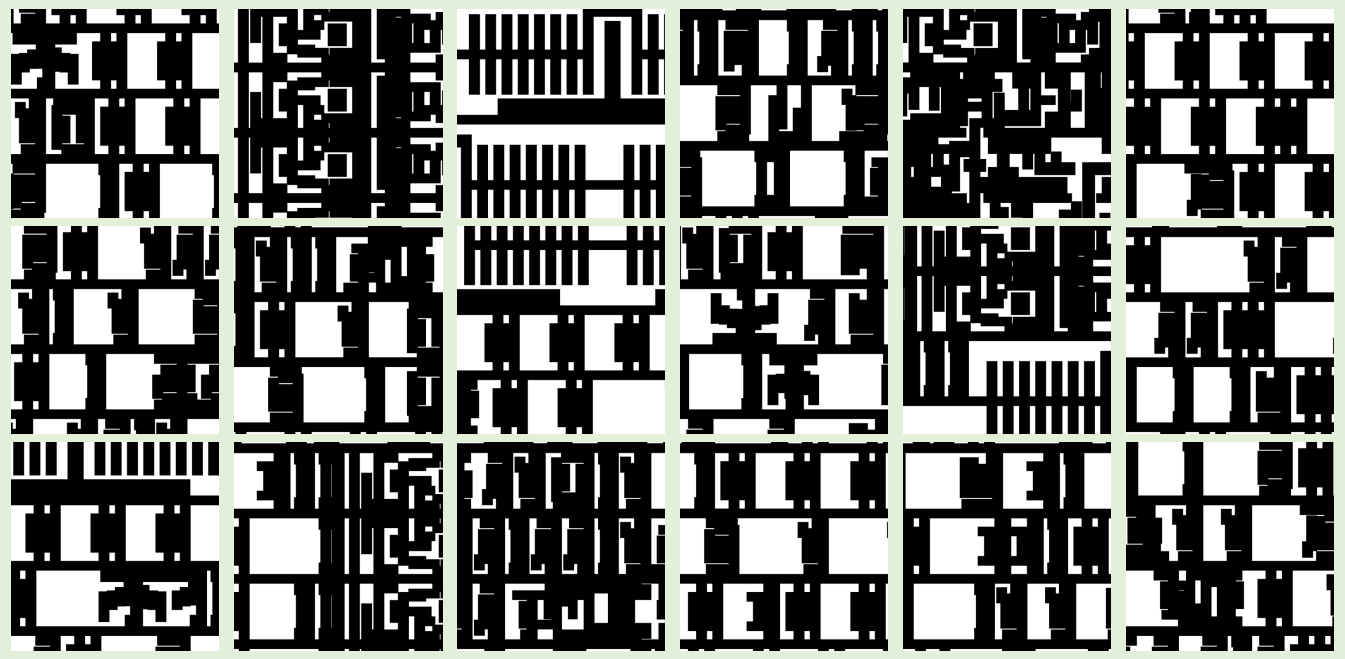


Figure 7: Examples of training patterns (white regions represent the metal shapes).

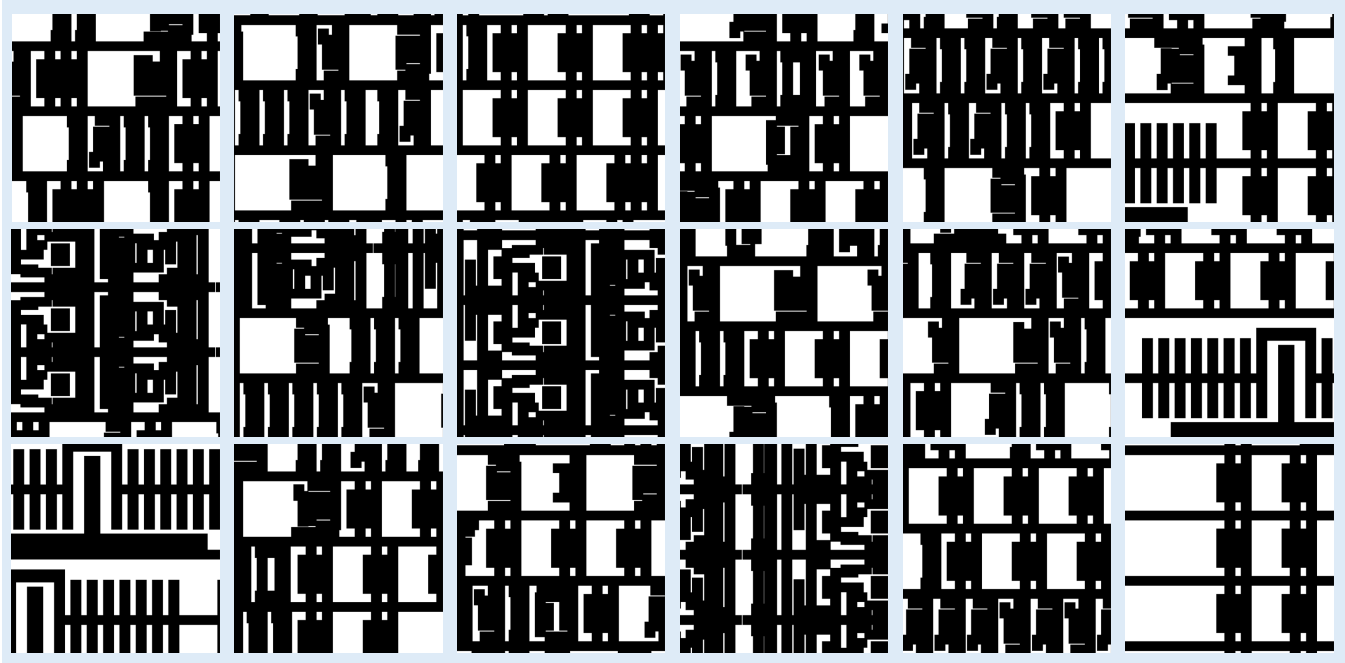


Figure 8: Examples of generated patterns (white regions represent the metal shapes).

one. This difference is defined as the pattern anomaly score. If a pattern x is realistic, the $score(x)$ should satisfy:

$$score(x) < T, \tag{9}$$

where T is a pre-defined threshold to determine the realism of a generated pattern.

For fair comparisons, we adopt the same pre-trained pattern style detection model in [24] to compute the anomaly scores of layout patterns generated by LayouTransformer. The legal patterns from the 100000 generated patterns are used to evaluate the pattern validity. We also cite the numbers of legal patterns (out of 100000) and the validity values obtained by [24] in Table 2. From this table, we can see that our LayouTransformer achieves the highest pattern

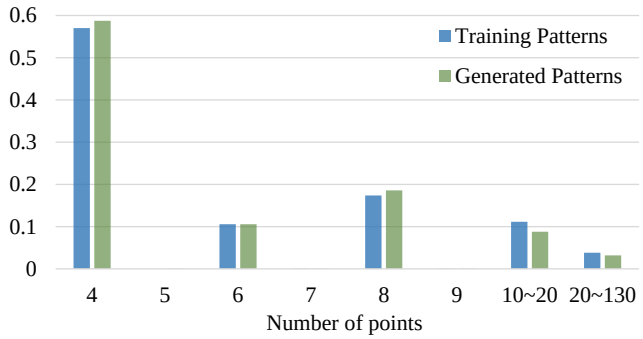


Figure 9: Comparison of the polygon shape distributions (indicated by the numbers of polygon vertexes) of the training set and the generated set.

validity with different thresholds, even approaching the validity of the real training patterns. The comparison between the training and generated patterns in Figure 7 and Figure 8 also shows that our generated patterns have high pattern validity from human's view.

6.4 Statistical Analysis of the Polygon Shape Distribution

The diversity of generated layout patterns plays a crucial role in various lithography design applications. Since the pattern complexity is defined as the number of scan lines subtracted by one along the x-axis and y-axis, the Shannon entropy of the pattern complexity only measures the difference among layout patterns. If the generated patterns with the high Shannon entropy of the pattern complexity are composed of only simple polygons, these generated patterns cannot improve lithography design applications, because the generated layout patterns need to have the same complexity of polygons as the training patterns.

Hence, we further evaluate the diversity of layout patterns generated by our method by comparing the polygon shape distribution of the generated patterns with that of the real training patterns. Specifically, the shape of a polygon is determined by its vertexes. For simplification, we count the number of vertexes of a polygon as its shape representation. Figure 9 shows the comparison of the polygon shape distribution in the training set and the generated set. We observe that the generated set has a similar distributions of polygon complexity to the training set. This further demonstrates the ability of LayouTransformer to generate diverse layout patterns whose shape distribution follows that of the training samples.

6.5 Generated Anomalous Patterns

Transformer-XL is employed to approximate the true distribution of the sequences of layout patterns based on the finite length-limited true sequences. This gives rise to the minor difference between the approximate distribution and the true distribution although Transformer-XL can well fit the training data. Since the generated pattern sequences are randomly sampled from the approximate distribution, some anomalous patterns are difficult to avoid.

These anomalous patterns can be divided into two categories. One is that a sequence between a pair [*SOP*] and [*EOP*] tokens cannot form a complete polygon with only horizontal and vertical edges. Figure 10 shows some such patterns. The proportion of this kind of anomalous patterns in 100000 generated patterns is only 0.48%. These incomplete polygons with only can be easily checked. Another category is that different polygons in a layout (partially) overlap spatially as shown in Figure 11. These layout patterns seriously violate the DRC rules. Their proportion in 100000 generated patterns is 7.9%. If we want to recycle these illegal patterns, the polygons can be adjusted according to the linear programming algorithm with the constraints of DRC rules [17]. LegalGAN can also be designed to reduce the number of overlapping polygons. These methods will be our further research for improvement. In this paper, we consider these anomalous patterns as illegal. As reported in Table 1, there are 89726 legal layout patterns out of 100000 generated layout patterns.

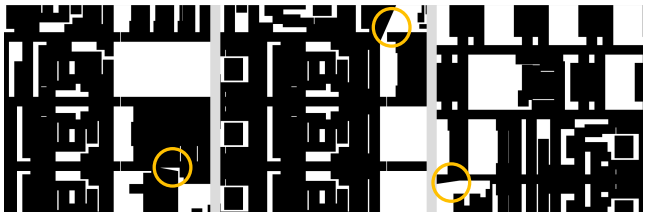


Figure 10: Some layout patterns with incomplete polygons.

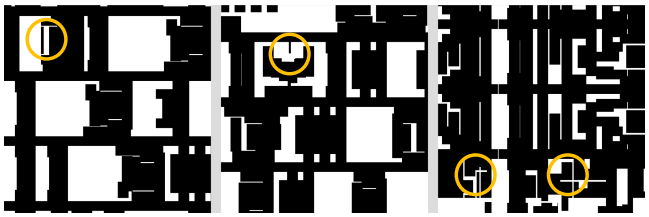


Figure 11: Examples of partially overlapped polygons.

7 CONCLUSION

In this work, we make the first attempt at learning the generation of layout patterns from the perspective of sequential modeling. To this end, we propose a novel and efficient sequential pattern representation to explicitly encode the shapes and layouts of patterns. This representation method is lossless, highly structured, and has high encoding efficiency. Also, we develop LayouTransformer based on Transformer which is a powerful architecture for sequence modeling and achieves great progress in language generation. LayouTransformer regards the design rules behind the training patterns as some kind of grammar rules. Compared to existing learning-based generative methods that typically generate irregular patterns due to pixel-level modeling, our LayouTransformer can well capture the long-term dependencies among the vertexes and edges of polygons as well as the spatial relationships among different polygons, and generate diverse, realistic and DRC-clean patterns for lithography design applications. Extensive experiments show that our LayouTransformer significantly outperforms existing learning-based methods in terms of pattern diversity, legality, and validity, demonstrating its superiority.

ACKNOWLEDGEMENT

We thank Xiaopeng Zhang for extremely helpful discussion.

REFERENCES

[1] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "Mosaic: Mask optimizing solution with process window aware inverse correction," in *51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014.

[2] A. Hamouda, M. Bahnas, D. Schumacher, I. Graur, A. Chen, K. Madkour, H. Ali, J. Meiring, N. Lafferty, and C. McGinty, "Enhanced opc recipe coverage and early hotspot detection through automated layout generation and analysis," in *Optical Microlithography*, vol. 10147, 2017, pp. 223–231.

[3] B. Jiang, H. Zhang, J. Yang, and E. F. Y. Young, "A fast machine learning-based mask printability predictor for opc acceleration," in *the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 412–419.

[4] J. Kuang, W.-K. Chow, and E. F. Y. Young, "A robust approach for process variation aware mask optimization," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 1591–1594.

[5] R. Chen, W. Zhong, H. Yang, H. Geng, X. Zeng, and B. Yu, "Faster region-based hotspot detection," in *56th ACM/IEEE Design Automation Conference (DAC)*, 2019.

[6] H. Yang, Y. Lin, B. Yu, and E. F. Y. Young, "Lithography hotspot detection: From shallow to deep learning," in *30th IEEE International System-on-Chip Conference (SOCC)*, 2017, pp. 233–238.

[7] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, pp. 1175–1187, 2019.

[8] H. Zhang, B. Yu, and E. F. Young, "Enabling online learning in lithography hotspot detection with information-theoretic feature optimization," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016.

[9] J. Kuang, J. Ye, and E. F. Young, "Simultaneous template optimization and mask assignment for dsa with multiple patterning," in *21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016, pp. 75–82.

[10] J. Kuang and E. F. Y. Young, "An efficient layout decomposition approach for triple patterning lithography," in *50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013.

[11] W. Ye, M. B. Alawieh, Y. Lin, and D. Z. Pan, "Lithogan: End-to-end lithography modeling with generative adversarial networks," in *56th ACM/IEEE Design Automation Conference (DAC)*, 2019.

[12] G. R. Reddy, K. Madkour, and Y. Makris, "Machine learning-based hotspot detection: Fallacies, pitfalls and marching orders," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

[13] G. R. Reddy, C. Xanthopoulos, and Y. Makris, "Enhanced hotspot detection through synthetic pattern generation and design of experiments," in *IEEE 36th VLSI Test Symposium (VTS)*, 2018.

[14] W. Ye, Y. Lin, M. Li, Q. Liu, and D. Z. Pan, "Lithoroc: Lithography hotspot detection with explicit roc optimization." Association for Computing Machinery, 2019.

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

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[17] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Deepattern: Layout pattern generation with transforming convolutional auto-encoder," in *56th ACM/IEEE Design Automation Conference (DAC)*, 2019.

[18] F. E. Gennari and Y.-C. La, "Topology design using squish patterns," *US Patent 8,832,621.*, 2014.

[19] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, 2020.

[20] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, J. Burstein, C. Doran, and T. Solorio, Eds., 2019, pp. 4171–4186.

[21] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 5754–5764.

[22] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," in *the 57th Conference of the Association for Computational Linguistics*, 2019, pp. 2978–2988.

[23] P. Kareem, Y. Kwon, and Y. Shin, "Layout pattern synthesis for lithography optimizations," *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 2, pp. 283–290, 2020.

[24] X. Zhang, J. Shiely, and E. F. Young, "Layout pattern generation and legalization with generative learning models," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020.

[25] P. Kareem and Y. Shin, "Synthesis of lithography test patterns using machine learning model," *IEEE Transactions on Semiconductor Manufacturing*, vol. 34, no. 1, pp. 49–57, 2021.

[26] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[27] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[29] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The curious case of neural text degeneration," in *8th International Conference on Learning Representations*, 2020.

[30] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.