DiffPattern: Layout Pattern Generation via Discrete Diffusion

Zixiao Wang[1*] Yunheng Shen[2*], Wenqian Zhao[1], Yang Bai[1], Guojin Chen[1], Farzan Farnia[1], Bei Yu[1]
[1]Chinese University of Hong Kong [2] Tsinghua University

***Abstract*—Deep generative models dominate the existing literature in layout pattern generation. However, leaving the guarantee of legality to an inexplicable neural network could be problematic in several applications. In this paper, we propose DiffPattern to generate reliable layout patterns. DiffPattern introduces a novel diverse topology generation method via a discrete diffusion model with compute-efficiently lossless layout pattern representation. Then a white-box pattern assessment is utilized to generate legal patterns given desired design rules. Our experiments on several benchmark settings show that DiffPattern significantly outperforms existing baselines and is capable of synthesizing reliable layout patterns.**

I. Introduction

Reliable Very-Large-Scale Integration (VLSI) layout pattern libraries are the foundation of various designs for manufacturability (DFM) research, including perfection of design rules, Optical Proximity Correction (OPC) recipes [1], lithography simulation [2], [3], layout hotspot detection [4], and so on. With the rapidly growing requirement for layout patterns in machine-learning-based lithography design applications, building a practical large-scale pattern library could be highly time-consuming due to the long logic-to-chip design cycle.

Various rule-based and learning-based layout pattern generation methods have been proposed to address this issue. Early rule-based methods [5], [6] first augment a set of predefined basic units by simple enhancement technology, *e.g.,* flipping and rotation. Then they randomly select several units and splice them together. However, patterns generated in this way lack diversity and are limited in quantity. Recently, learning-based generative methods [7]–[9] have shown the ability to produce a large number of diverse layout patterns. Among them, pixel-based methods [7], [8] regard the pattern generation problem as a binary image generation task. A lossless pattern representation, *Squish Pattern*, is proposed by [10] to reduce computational costs. Squish Pattern splits a large layout pattern into a significantly smaller binary topology matrix and two geometric vectors. Pixel-based methods synthesize a topology matrix with continual values and threshold it to get a new binary topology matrix, which wastes computation and hurts model capacity. Also, the sequential-based method in [9] models a layout pattern as a sequence of polygons, which is further decomposed into a sequence of vertices and directed edges. To get a new layout pattern, [9] generates a polygon sequence and translates it into a layout pattern. For both kinds of them, the generative model learns a latent regularization from the training set to avoid producing illegal layout patterns that violate the design rules. We argue that the implicit constraint learned from the training set may be neither flexible nor reliable. Apart from the inconvenience that we need to train a new model on a specific large-scale dataset that follows the new design rules when the design rule changes, a considerable ratio of generated patterns violate the design rules in these methods.

In this paper, we propose DiffPattern, a pixel-based practical layout pattern generation framework that consists of three main components: a) Inspired by the great success of diffusion models [11], [12], we approach the generation of topology as a denoising task for a random noise image. We utilize a discrete diffusion model to predict the noise that should be removed at every step. During the training and sampling procedure, each entry of the image tensor can be expressed by a discrete state in a finite state space, $\{0,1\}$ for example, and there is no need to manually set a threshold on a continual output range as previous work does. The naturally discrete output provides a reasonable regularization that avoids meaningless overfitting on how to produce a discrete-like output since all the pixels in the training samples are discrete in their values and position. b) To further enlarge the information density of patterns and reduce computation costs, DiffPattern adapts the idea of Squish Pattern Representation and pushes it one step forward. A kind of lossless layout pattern representation method called *Deep Squish Pattern* is proposed. Deep Squish Pattern folds a topology matrix into a topology tensor, which shrinks the input size and extends the channel dimension. Due to the fact that the efficiency of the existing diffusion model is more sensitive to the input size [13] and significantly less to the number of input channels, Deep Squish Pattern provides an easy-to-compute solution and can be efficiently applied to other pixel-based pattern generation methods. c) With newly generated topology matrices, we aim to assign geometric vectors to them and restore legal layout patterns. We develop a nonlinear system that can figure out the legal solution for each topology matrix and can be easily adjusted to every design rule. Empowered by the interpretable pattern assessment strategy, DiffPattern achieves a notable 100% legality rate on generated layout patterns in our setting.

This work is supported by The Research Grants Council of Hong Kong SAR (No. CUHK14208021).

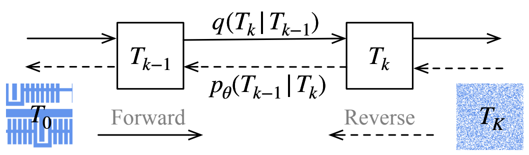


Fig. 1 Illustration of denoising diffusion process.

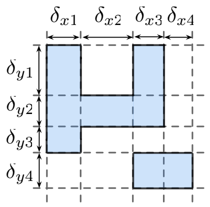
Our main contributions can be summarized as follows:

1) We develop a novel layout pattern generation method based on discrete denoising for synthesizing layout topology.
2) We propose a lossless layout pattern representation strategy, Deep Squish Pattern, which accelerates pixel-based layout pattern generation schemes.
3) We utilize a nonlinear system for pattern assessment where the system provides a white-box method to legalize the layout patterns.
4) We extensively evaluate our methodology on benchmark datasets showing that DiffPattern can achieve state-of-the-art (SOTA) performance.

II. Preliminaries

A. Diffusion Models

Diffusion models refer to denoising diffusion probabilistic models [11], which have demonstrated great potential in generating high-quality images. A diffusion model defines a Markov chain to represent the sample forward and reverse diffusion process, as shown in Fig. 1. In the forward diffusion process, the diffusion model produces a sequence



Topology: $\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$

Geometry: $\Delta_x = [\delta_{x1}, \delta_{x2}, \delta_{x3}, \delta_{x4}]$
$\Delta_y = [\delta_{y1}, \delta_{y2}, \delta_{y3}, \delta_{y4}]$

Fig. 2 Squish Pattern Representation.

of noisy samples $\boldsymbol{T}_1, ..., \boldsymbol{T}_K$ by iteratively adding Gaussian noise to the original sample $\boldsymbol{T}_0$ in K steps, where the noise level is controlled by a variance schedule $\{\beta_k \in (0,1)\}_{k=1}^{K}$:

$$\boldsymbol{q}\left(\boldsymbol{T}_k | \boldsymbol{T}_{k-1}\right) := \mathcal{N}\left(\boldsymbol{T}_k; \sqrt{1-\beta_k}\boldsymbol{T}_{k-1}, \beta_k \boldsymbol{I}\right). \quad (1)$$

When K is large, the noisy sample at the final iteration $\boldsymbol{T}_K$ will have an almost Gaussian distribution, which inspires the diffusion model to construct a reverse diffusion process to produce fresh data samples from randomly drawn Gaussian noise. The goal of the reverse diffusion process is to learn the distribution of inverting the explained forward diffusion process, which will be used to generate fresh data samples from a Gaussian noise input. However, a proper inversion of the distribution requires an expressive model, and therefore we use a deep neural network with learnable parameters $\boldsymbol{\theta}$ to learn the distribution,

$$\boldsymbol{p}_{\boldsymbol{\theta}}\left(\boldsymbol{T}_{k-1} | \boldsymbol{T}_k\right) := \mathcal{N}\left(\boldsymbol{T}_{k-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}\left(\boldsymbol{T}_k, k\right), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}\left(\boldsymbol{T}_k, k\right)\right). \quad (2)$$

Similar to variational autoencoders (VAEs), the training of diffusion models aims to maximize the log-likelihood function by optimizing the variational lower bound.

$$L_{\mathrm{VLB}} = D_{\mathrm{KL}}\left(\boldsymbol{q}\left(\boldsymbol{T}_K | \boldsymbol{T}_0\right) \parallel \boldsymbol{p}_{\boldsymbol{\theta}}\left(\boldsymbol{T}_K\right)\right) + \sum_{k=2}^{K} L_k - \log \boldsymbol{p}_{\boldsymbol{\theta}}\left(\boldsymbol{T}_0 | \boldsymbol{T}_1\right), \quad (3)$$

where $L_k = D_{\mathrm{KL}}\left(\boldsymbol{q}\left(\boldsymbol{T}_{k-1} | \boldsymbol{T}_k, \boldsymbol{T}_0\right) \parallel \boldsymbol{p}_{\theta}\left(\boldsymbol{T}_{k-1} | \boldsymbol{T}_k\right)\right)$, D_{KL} is the KL divergence, and the term $\boldsymbol{q}\left(\boldsymbol{T}_{k-1} | \boldsymbol{T}_k, \boldsymbol{T}_0\right)$ can be derived a closed form of Gaussian distribution according to Equation (1) and Bayes' theorem.

After training the diffusion model, we can simply sample from a standard Gaussian distribution and remove the noise in an iterative fashion according to Equation (2) that follows the reverse diffusion process, resulting in new samples that follow the original sample distribution.

B. Squish Pattern Representation

A typical layout pattern consists of a stack of polygons and is information-sparse, which leads to unnecessary computational costs and additional overfitting risk for neural network methods. The squish pattern [10] is a lossless and efficient representation method which encodes a layout into pattern topology matrix and geometric information Δ_x, Δ_y, as shown in Fig. 2. The layout is split into grids by a set of scan lines that walk along the edges of polygons. The interval length of every adjacent scan line pair is stored in the Δ vectors. Every entry of the topology matrix is either zero or one, which indicates shape or zeros respectively. Every squish pattern is extended to a square with fixed side length by the method introduced in [14].

C. Problem Formulation

The diversity of generated patterns is a critical metric for the layout pattern generation task. As introduced in [7], the complexity of a layout pattern is defined 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. Then we have,

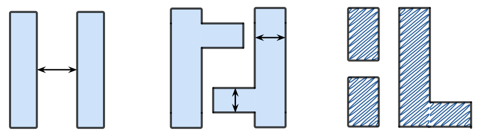


Fig. 3 Illustration of design rules.

Definition 1. *The diversity of the patterns library, denoted by H, is defined as the Shannon Entropy of the distribution of the pattern complexities as follows:*

$$H = -\sum_i \sum_j P(c_{xi}, c_{yj}) \log P(c_{xi}, c_{yj}), \quad (4)$$

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

In a typical case, a greater pattern diversity H indicates that the library contains more widely distributed patterns.

Based on the related works [8], [9], the generated patterns should satisfy several pre-defined design rules of IC layout. As illustrated in Fig. 3, 'Space' represents the distance between two adjacent polygons. 'Width' measures the size of a shape in one direction. And 'Area' denotes the area of a polygon. Based on these geometric measurements, we have,

Definition 2 (Pattern Legality)**.** *We call a layout pattern legal if the layout pattern is DRC-clean, given the design rules.*

Based on the above evaluation metrics, the pattern generation problem can be formulated as follows,

Problem 1 (Pattern Generation)**.** *Given a set of design rules and existing patterns, the objective of pattern generation is to synthesize a legal pattern library such that the pattern diversity of the layout patterns in the library are maximized.*

III. Framework

A. Overview of DiffPattern

As illustrated in Fig. 4, our framework consists of three phases: (1) Deep Squish Pattern Representation. Given a set of layout patterns, we first extract their deep squish pattern representation in which every layout pattern is decomposed into a topology tensor $\boldsymbol{T}$ and two geometric vectors $\boldsymbol{\Delta}_x$ and $\boldsymbol{\Delta}_y$. (2) Topology Tensor Generation. We subsequently feed the extracted topology tensors $\boldsymbol{T}_0$ into a diffusion model. By gradually adding noise to $\boldsymbol{T}$ with fixed probabilities $q(\boldsymbol{T}_k | \boldsymbol{T}_{k-1})$, the model learns how to reverse this K-step process. To synthesize a new topology tensor $\hat{\boldsymbol{T}}$, we randomly sampled a noise topology $\boldsymbol{T}_K$, and every entry in $\boldsymbol{T}$ jumps between finite states with estimated probability $p_\theta(\boldsymbol{T}_{k-1} | \boldsymbol{T}_k)$ until getting a reasonable topology tensor $\hat{\boldsymbol{T}}_0$. (3) 2D Legal Pattern Assessment. Given the generated topology tensors $\hat{\boldsymbol{T}}$, we established an explainable nonlinear system to figure out the proper geometric vectors for each topology tensor according to the *Design Rules*.

B. Deep Squish Pattern Representation

As we have discussed in Section II-B, the squish pattern is a lossless representation of the layout pattern, and the topology matrix can be treated as a one-channel 2D binary mask, as shown in Fig. 5 (left). However, the information density of each pixel is still not satisfactory given that the training/inference efficiency of diffusion methods is more sensitive to the size of input images but significantly less on the number of states of each pixel. To address this issue, we propose a novel pattern

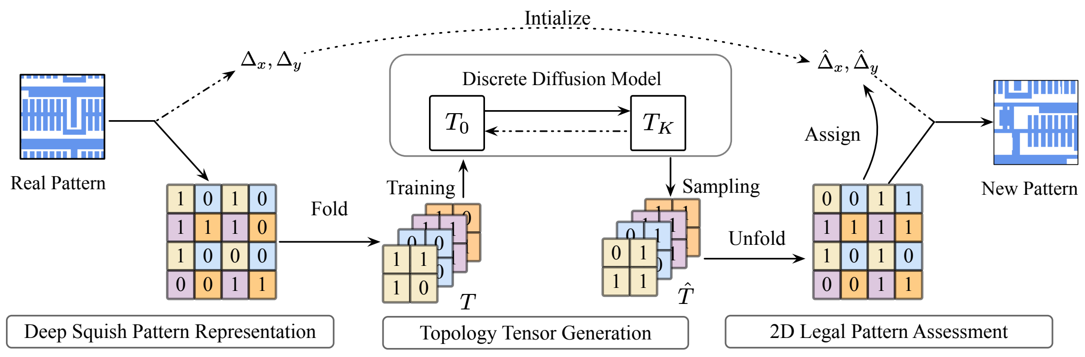


Fig. 4 An illustration of the Diffpattern framework for reliable layout pattern generation.

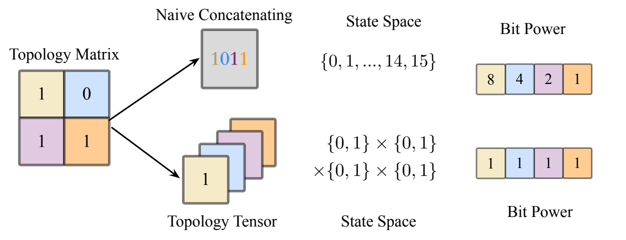


Fig. 5 An illustration of Deep Squish Pattern Representation. The Topology Tensor is a lossless and compact representation of the topology matrix. And the Naive Concatenating brings unbalanced power to each bit and an exponentially increasing state space.

representation method, *Deep Squish Pattern,* to attain a more compact pattern representation.

Here we use a running example, as shown in Fig. 5. There is a topology matrix with four (2×2) adjacent pixels. Each pixel is assigned with either zero or one to indicate shape or space, respectively. A simple idea to enlarge the information density is to encode the bits from multiple pixels into one pixel. However, directly concatenating every bit from different pixels and forming a new state number, *e.g.,* using 0-15 to represent 16 different states of (2×2) adjacent pixels, will bring unbalanced power to each position and may import numerical instability to the network when the bit count is large. For example in the 4×4 case, the first bit gets a power of 2^{15} while the last bit only gets a power of 1. Furthermore, the count of states increases exponentially with the count of bits.

Since a state space with 16 (2^4) discrete states can be expressed by a permutation of four subspaces with two states. Instead of encoding different positions with different power, we assign equivalent weight to each position by folding the squish topology matrix into a topology tensor $\boldsymbol{T}$ with multiple channels. In the folding step, a patch with the size of $\sqrt{C} \times \sqrt{C}$ from the topology matrix is transferred to a point with C channels in the tensor $\boldsymbol{T}$. C is a hyperparameter, which is chosen by considering the trade-off between local information density and the input size. Deep Squish Pattern representation enlarges the practical receptive field of the model and is naturally suitable for pixel-based machine learning methods. Once the generation procedure ends, we can also equivalently recover the original topology matrix by flattening the topology tensor.

C. Topology Tensor Generation

Once we have encoded the existing layout pattern efficiently by using deep squish pattern representation, we aim to learn the distribution of existing topology tensors and generate new ones with reasonable topology attributes. Let $\boldsymbol{T}_0 \in \{0,1\}^{C\times M\times M}$ be a topology tensor extracted from existing patterns. The naive idea is that treat the binary tensor as a grayscale image, learn the distribution through a diffusion model introduced in Section II-A and clip the generated topology into a binary one by setting a threshold on it as previous pixel-based pattern generation methods [7], [8] do. The problem of learning how to generate discrete values (zero and one in our case) at every entry is left to the network. However, we argue that requesting the network learn a discrete output in a continual state space from the training set is a waste of model representation ability. A more elegant way is to generate discrete output naturally.

Discrete Diffusion Model. Unlike the traditional diffusion model utilized in the computer vision domain, we aim to synthesize the topology of layout pattern, where every entry in the topology belongs to a discrete state. We have made several key modifications to enhance the diffusion model and synthesize discrete topology patterns directly. To derive them, we first re-formulate the problem.

At the k-th of K diffusion step, $x_k \in \{0,1\}$ is an entry in the topology tensor $\boldsymbol{T}$. In the discrete diffusion model, a transition probability matrix $[\boldsymbol{Q}_k]_{ij} = q(x_k = j|x_{k-1} = i)$ is defined to describe the state transition probability for each $\boldsymbol{x}$ at the k-th diffusion step,

$$\boldsymbol{q}\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}\right) := \mathrm{Cat}\left(\boldsymbol{x}_k; \boldsymbol{p} = \boldsymbol{x}_{k-1}\boldsymbol{Q}_k\right), \tag{5}$$

where $\boldsymbol{x}_k$ is the one-hot version of the entry x_k, $\mathrm{Cat}(\boldsymbol{x}|\boldsymbol{p})$ is a categorical distribution over the row vector $\boldsymbol{x}$ with probabilities given by the row vector $\boldsymbol{p}$, and $\boldsymbol{x}_{k-1}\boldsymbol{Q}_k$ can be understood as a row vector-matrix product. The $\boldsymbol{Q}_k$ is applied to each entry in the topology tensor independently and $\boldsymbol{q}$ factorizes over these higher dimensions as well. The proposed deep squish pattern representation is customized for this discrete diffusion process, since the size of transition matrix $\boldsymbol{Q}$ increases with the count of states of each pixel. Every entry x in the topology tensor owns only two states. On the other side, in the reverse diffusion process, the neural network aims to predict the categorical distribution probability $\boldsymbol{p}_\theta(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k)$ over each entry to recover the original tensor.

The choice of transition probability matrix $\boldsymbol{Q}_k$ is critical, which should ensure the forward process $\boldsymbol{q}(\boldsymbol{x}_k|\boldsymbol{x}_0)$ converging to a known stationary distribution when k becomes large. A uniform stationary distribution is a natural choice in topology tensor generation, which means given any $\boldsymbol{x}_0$, the distribution of every entry $\boldsymbol{x}_k$ should follows,

$$\boldsymbol{q}(\boldsymbol{x}_k|\boldsymbol{x}_0) \to [0.5, 0.5]\,, \text{ when } k \to K. \tag{6}$$

So we design a doubly stochastic matrix $\boldsymbol{Q}_k$ with strictly positive entries for topology denoising diffusion process,

$$\boldsymbol{Q}_k = \begin{bmatrix} 1-\beta_k & \beta_k \\ \beta_k & 1-\beta_k \end{bmatrix}, \tag{7}$$

where $\beta_k \in (0,1)$ is the hyperparameter controlling the noise level. In order to ensure that the model can learn the original sample distribution more finely and quickly reach a stable distribution, we follow the classical setting in previous works [11], [15]. We set a smaller noise in the early diffusion step and a larger noise in the later step. Specifically, we use a linearly increasing schedule for β_k:

$$\beta_k = \frac{(k-1)(\beta_K - \beta_1)}{K-1} + \beta_1, \ k = 1, ..., K, \tag{8}$$

where β_1 and β_K are hyperparameters.

Training Diffusion Model. To training the discrete diffusion model for topology tensor generation, the training objective at step k is to minimize the loss function,

$$L = D_{\mathrm{KL}}\left(\boldsymbol{q}\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \boldsymbol{x}_0\right) \parallel \boldsymbol{p}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k\right)\right) - \lambda \log \boldsymbol{p}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_0|\boldsymbol{x}_k\right), \tag{9}$$

where λ is a hyperparameter to balance the loss terms.

Given a topology tensor $\boldsymbol{T}_0$, we randomly sample a target step k from 1 to K firstly, and expect to get the noisy sample $\boldsymbol{T}_k$. Fortunately, we can explicitly derive that $\boldsymbol{x}_k$ obeys the following categorical distribution:

$$\boldsymbol{q}\left(\boldsymbol{x}_k|\boldsymbol{x}_0\right) = \mathrm{Cat}\left(\boldsymbol{x}_k; \boldsymbol{p} = \boldsymbol{x}_0\overline{\boldsymbol{Q}}_k\right), \tag{10}$$

where $\overline{\boldsymbol{Q}}_k = \boldsymbol{Q}_1\boldsymbol{Q}_2\ldots\boldsymbol{Q}_k$. Then, we can directly sample from the above distribution to obtain $\boldsymbol{T}_k$, instead of adding noise k times.

After sampling $\boldsymbol{T}_k$, we feed it into the neural network with the embedding of the time step k. The neural network will predict the logits of the posterior distribution $\boldsymbol{p}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_0|\boldsymbol{x}_k\right)$, and then $\boldsymbol{p}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k\right)$ can be calculated as following:

$$\boldsymbol{p}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k\right) = \sum_{\widetilde{\boldsymbol{x}}_0} \boldsymbol{q}\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \widetilde{\boldsymbol{x}}_0\right)\boldsymbol{p}_{\boldsymbol{\theta}}\left(\widetilde{\boldsymbol{x}}_0|\boldsymbol{x}_k\right), \tag{11}$$

where the term $\widetilde{\boldsymbol{x}}_0$ will visit every possible state of $\boldsymbol{x}_0$. And $\boldsymbol{q}\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \widetilde{\boldsymbol{x}}_0\right)$ has a closed form according to Equation (5) and Bayes' theorem,

$$\boldsymbol{q}\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \boldsymbol{x}_0\right) = \mathrm{Cat}\left(\boldsymbol{x}_{k-1}; \boldsymbol{p} = \frac{\boldsymbol{x}_k\boldsymbol{Q}_k^\top \odot \boldsymbol{x}_0\overline{\boldsymbol{Q}}_{k-1}}{\boldsymbol{x}_0\overline{\boldsymbol{Q}}_k\boldsymbol{x}_k^\top}\right), \tag{12}$$

where $\odot$ is a pixel-wise multiplication.

So far, all items in the loss function have been obtained, and the diffusion model can be trained by the commonly used gradient descent method.

Generating Deep Squish Pattern. Once the training phase ends, we can synthesize fresh topology by sampling noise topology $\boldsymbol{T}_K$ from the stationary distribution, a random uniform distribution, and then gradually removing the predicted noise from it in the reverse procedure. The sampling process can be expressed by,

$$p_\theta(\hat{\boldsymbol{T}}_0|\boldsymbol{T}_K) = p_\theta(\hat{\boldsymbol{T}}_0|\boldsymbol{T}_1)\prod_{k=2}^{K} p_\theta(\boldsymbol{T}_{k-1}|\boldsymbol{T}_k), \tag{13}$$

where $\boldsymbol{T}_k$ is the estimated pattern topology at step k and $\hat{\boldsymbol{T}}_0$ is the newly sampled topology tensor. The denoising procedure is illustrated in Fig. 6. The generated topology tensor $\hat{\boldsymbol{T}}_0$ is naturally a binary one where each entry equals either zero or one. When the sampling ends, we flatten the $\hat{\boldsymbol{T}}_0$ for Legal Pattern Assessment.

Topology Pre-filter. We conduct a rule-based topology pre-processing here to filter out invalid topology, *e.g.,* Bow-tie, according to domain knowledge. Thanks to the high-quality topologies generated by our discrete diffusion model, only less than 0.1% of generated topologies are filter-out by the pre-filter in our settings.

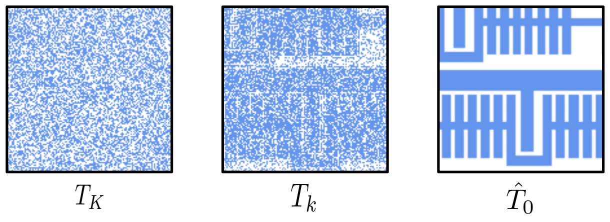


Fig. 6 An illustration of the (flattened) samples from our Discrete Diffusion Model.

D. 2D Legal Pattern Assessment

Once the squish pattern generation is finished, we need the legal Δ_xs and Δ_ys of all generated topologies to generate DRC-clean layout patterns. The decomposition of topology generation and legal pattern assessment brings flexibility to DiffPattern when the design rules change, as we further discussed in Section IV-C. Instead of using a black-box deep-learning-based method as previous works [8], [9] do, we use a white-box method to solve the problem. We first list all constraints for each generated topology according to the design rules introduced in Fig. 3 and then formulate a nonlinear system combining all of them, as in Formula (14),

$$\begin{cases} \delta_{xi}, \delta_{yj} > 0, & \forall \delta_{xi}, \delta_{yj}; \\ \sum \delta_{xi} = \sqrt{C}M, \quad \sum \delta_{yj} = \sqrt{C}M; & \\ \sum_{i=a}^{b} \delta_i \geq Space_{min}, & \forall (a,b) \in Set_S; \\ \sum_{i=a}^{b} \delta_i \geq Width_{min}, & \forall (a,b) \in Set_W; \\ \sum \delta_{xi}\delta_{yj} \in [Area_{min}, Area_{max}], & \forall \text{ Polygon}; \end{cases} \tag{14}$$

where $Space_{min}$ and $Width_{min}$ are the lower bound of 'Space' and 'Width'. $\sqrt{C}M \times \sqrt{C}M$ is the shape of topology matrix. $Area_{min}$ and $Area_{max}$ define the legal area range of each polygon in the pattern. All the constants are pattern-independent and given by design rules. Both Set_S and Set_W are pattern-dependent and indicate which pair of scan lines is constrained by design rules on 'Space' and 'Width', respectively.

Note that the nonlinear system in Equation (14) can be efficiently solved with vast nonlinear programming algorithms or numerical methods, and the solution is usually not unique. Every solution of Δ_x and Δ_y together with the associated topology formulates a complete squish pattern representation. As further detailed in Section IV-C, DiffPattern can easily synthesize a large number of legal layout patterns from a single topology with the given design rules. Theoretically, there are cases where no legal solution is figured out in a limited time. Although it never happens in our experiments (more than 1.0×10^8 times attempts), we can simply remove these unsolvable cases from the generated topology set to avoid synthesizing illegal patterns. In most cases, the choice of the initial value for the nonlinear programming algorithms may have a minor impact on pattern diversity and legality. In practice, we randomly choose a pair of existing geometric vectors from datasets as the start point, which will empirically accelerate the convergence of nonlinear programming algorithm.

IV. Experimental Results

A. Experimental Setup

Datasets. We follow previous work [8], [9] to obtain the dataset of small layout pattern images with the size of 2048 × 2048 nm^2 by splitting a 400 × 160 μm^2 layout map from ICCAD contest 2014. The size of extracted topology tensor is fixed as 16×32×32 with $C = 16$ in Deep Squish Pattern Representation. 3000 images are randomly choosed as test set, while others are used for training.

TABLE I Comparison on pattern diversity and legality. All results are copied from [9] and [8]. 'Real Patterns' refer to the whole dataset (Training set + Test set). '-' refer to Not Applicable.

Set/Method	Generated Topology	Generated Patterns		Legal Patterns	
		Patterns	Diversity (↑)	Legality (↑)	Diversity (↑)
Real Patterns	-	-	-	13869	10.777
CAE [7]	100000	100000	4.5875	19	3.7871
VCAE [8]	100000	100000	**10.9311**	2126	9.9775
CAE+LegalGAN [8]	100000	100000	5.8465	3740	5.8142
VCAE+LegalGAN [8]	100000	100000	9.8692	84510	9.8669
LayouTransformer [9]	-	100000	10.532	89726	10.527
DiffPattern-S	100000	100000	10.815	**100000**	**10.815**
DiffPattern-L	100000	10000000	10.815	**10000000**	**10.815**

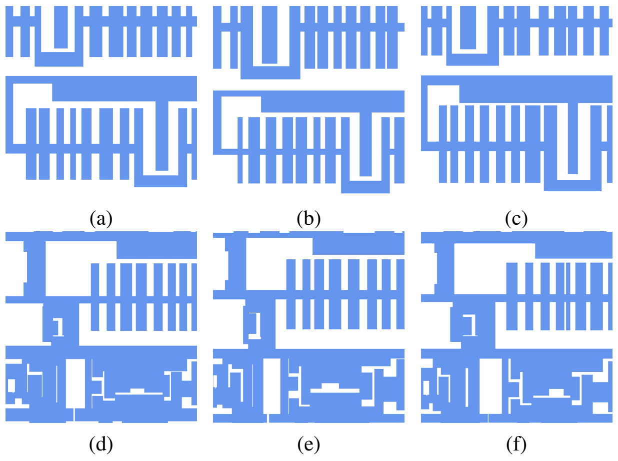


Fig. 7 Different layout patterns that are generated from a single topology with the same design rule.

Diffusion Model Configuration. Following the setting in previous works [11], [15], we use a U-Net [16] as our backbone in the discrete diffusion model to predict the posterior distribution during the reverse diffusion process. There are four feature map resolutions in the model, $[32\times32, 16\times16, 8\times8, 4\times4]$. Each resolution level has two convolutional residual blocks, where the numbers of convolution channels on the four resolutions are $[128, 256, 256, 256]$, respectively. A self-attention block is also placed between two convolutional blocks at the 16×16 resolution level. Moreover, the time step k is included in each residual block through the sinusoidal position embedding [17].

Training Details. We train the diffusion model for 0.5M iterations with a batch size of 128. The learning rate is 2×10^{-4}. And we use the Adam optimizer. Other hyperparameters are chosen as following: the dropout rate is set to 0.1, the grad clip is set to 1, and the loss coefficient λ is set to 0.001. We set diffusion timesteps $K = 1000$ to ensure that the forward diffusion process converges to the uniform stationary distribution within K steps. The noise schedule β_k is linearly increased from 0.01 to 0.5. The training procedure takes about 17 hours in total on 8 NVIDIA RTX 3090 GPUs.

B. Pattern Diversity and Legality

The diversity is calculated by Equation (4), and the legality is checked by a tool *Klayout* based on the design rules described in Section II-C.

To have a fair comparison with previous methods, we randomly synthesize 100000 typologies. We denote a version of DiffPattern as DiffPattern-S where we only assign one pair of geometric vectors to each generated topology in the phase of legal pattern assessment. However, DiffPattern is able to generate a large amount of legal patterns for each topology. To show the strong ability of our methodologies, we have another version denoted as DiffPattern-L, where we figure out one hundred different legal geometric vectors for each generated topology. Considering the size of topology and the large search space, the number of legal solutions for each topology is huge. We 'only' randomly take one hundred from them due to the time limitation.

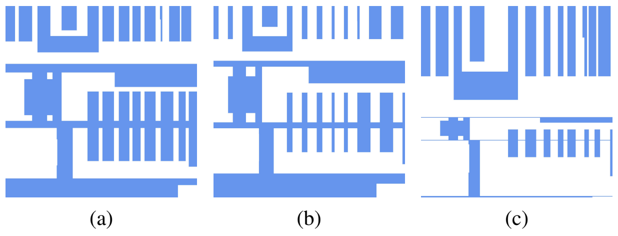


Fig. 8 Layout patterns that are generated from the same topology with different design rules: (a) Normal rule; (b) Larger $space_{min}$; (c) Smaller $Area_{max}$.

We compare our DiffPattern with several learning-based layout pattern generation methods. CAE [7] denotes a vanilla convolutional auto-encoder model. VCAE [8] utilizes a variational convolutional auto-encoder model. Both of them are pixel-based methods. LegalGAN [8] is a learning-based post-processing method that legalizes a newly generated topology by modifying it. LayouTransformer [9] is a sequential-based method. LayouTransformer applies a transformer-based model to synthesize new sequential representations of layout patterns without topology generation. As shown in TABLE I, thanks to the topology pre-filter and rule-based 2D legal pattern assessment, both DiffPattern-S and DiffPattern-L achieve perfect performance (*i.e.* 100%) under the metric of legality in the standard settings. With the well-designed topology generation method, DiffPatten also gets reasonable improvement (10.527→10.815) on the diversity of generated pattern compared with the previous best method, LayouTransformer. Furthermore, different from the previous methods, the legalization of DiffPattern mainly relies on the rule-based legal pattern assessment and topology pre-filter. When design rules change, it is easy for us to produce another batch of diverse patterns that satisfy the new design rules, without re-training the topology generation model. We detail the flexibility of our method in the next subsection.

C. Flexibility

A main advantage of DiffPattern is that the white-box 2D legal pattern assessment phase is quite flexible. We show two applications of this property here.

Generate Different Patterns from Single Topology. Given the design rules and the topology, the non-linear system in Equation (14) usually has a large number of legal solutions, *i.e.,* the geometric vector pairs. Every legal solution leads to a legal layout pattern, and these

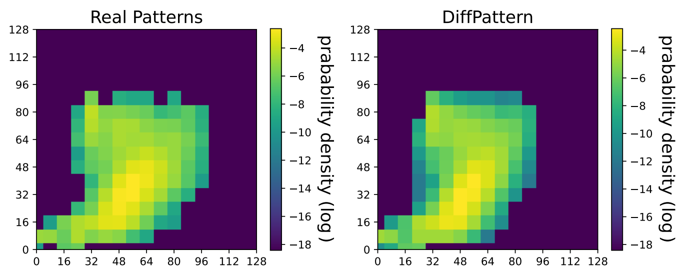


Fig. 9 An illustration of complexity distribution.

layout patterns share a common topology, which are useful in some downstream tasks. We show the examples where several layout patterns are generated from a single topology with different geometric vectors in Fig. 7.

Generate Legal Patterns with Different Design Rules. The legalization and topology generation are decoupled in DiffPattern, which allows us to generate legal layout patterns under different design rules without re-training the model. We show the examples where several layout patterns are generated from the same topology but with different design rules in Fig. 8.

D. Distribution of Complexity

Diversity is a critical metric of the quality of generated pattern library. As defined in Section II-C, diversity is the Shannon entropy of the distribution of pattern complexity, *i.e.,* the number of scan lines subtracted by a pattern along the x-axis and y-axis. We visualize the distribution of complexity in Fig. 9. The patterns generated by DiffPattern share a similar complexity distribution with real patterns. This visualization further demonstrates our ability to generate high-quality layout patterns.

E. Model Efficiency

Model efficiency is an essential statistic for layout generation methods. Since the topology generation and layout pattern assessment are decomposed in our model, we record the average time to sample a new topology and figure out a legal solution for Equation (14) separately with our implementation. The results are listed in TABLE II. We have mentioned in Section III-D that we randomly select a pair of existing geometric vectors to initialize the non-linear system for acceleration. We denote the version as *Solving-E*. We compare it with the version using a random initialization, *Solving-R*. Solving-E achieves an average of 2.30× acceleration.

F. Discussion on Validity

There is a metric named pattern validity proposed by previous work [8]. The evaluation of validity is based on an encoder-decoder model that is pre-trained on the training set. The basic idea is if the generated patterns share more similar features with patterns in the training set, these new patterns will get a better score from the validity metric. However, we argue that the meaning of the metric of validity is quite limited. One main motivation of the layout pattern generation task is to synthesize a large amount of diverse but legal layout patterns for downstream tasks, *e.g.,* hotpot detection or lithography simulation. In these situations, legal patterns that are dissimilar from existing patterns are preferred but they get worse scores from validity. What makes the metric worse is that the measurement of validity encourages the overfitting of the training set. For example, according to [8], [9], the generated patterns even receive a much higher score (65%→84%) than the patterns from the test set, which theoretically follows the same distribution as the training set. It is unreasonable to believe the method with the better score in validity is superior in quality. Therefore, we have not evaluated DiffPattern with this metric.

TABLE II Model efficiency of DiffPattern. We record the average processing time for each sample (topology or layout pattern) in our machine. One Nvidia RTX 3090 GPU is used for topology sampling and one Intel(R) Xeon(R) Gold 6326 CPU @ 2.90GHz is used to figure out the non-linear system in this table.

Phase/Method	Cost Time (s)	Acceleration
Sampling	0.544	N/A
Solving-R	0.269	1.00×
Solving-E	0.117	2.30×

V. Conclusion

We aim to synthesize diverse and legal layout patterns, and propose a practical method named DiffPattern. Based on the given design rules our approach allows us to stably generate theoretically infinite legal layout patterns. The experiment results show that DiffPattern outperforms previous SOTA methods by a large margin. Meanwhile, DiffPattern is easy to extend and can be used on more complex pattern generation scenarios. In future work, we hope to extend our method to a more generic layout pattern generation approach.

References

[1] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "Mosaic: Mask optimizing solution with process window aware inverse correction," in *DAC*. IEEE, 2014, pp. 1–6.
[2] J. Kuang and E. F. Young, "An efficient layout decomposition approach for triple patterning lithography," in *DAC*. IEEE, 2013, pp. 1–6.
[3] B. Yu, K. Yuan, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," *TCAD*, vol. 34, no. 3, pp. 433–446, 2015.
[4] R. Chen, W. Zhong, H. Yang, H. Geng, X. Zeng, and B. Yu, "Faster region-based hotspot detection," in *DAC*, 2019, pp. 1–6.
[5] G. R. Reddy, C. Xanthopoulos, and Y. Makris, "Enhanced hotspot detection through synthetic pattern generation and design of experiments," in *VTS*. IEEE, 2018, pp. 1–6.
[6] W. Ye, Y. Lin, M. Li, Q. Liu, and D. Z. Pan, "Lithoroc: lithography hotspot detection with explicit roc optimization," in *DAC*, 2019, pp. 292–298.
[7] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Deepattern: Layout pattern generation with transforming convolutional auto-encoder," in *DAC*, 2019, pp. 1–6.
[8] X. Zhang, J. Shiely, and E. F. Young, "Layout pattern generation and legalization with generative learning models," in *ICCAD*. IEEE, 2020, pp. 1–9.
[9] L. Wen, Y. Zhu, L. Ye, G. Chen, B. Yu, J. Liu, and C. Xu, "Layoutransformer: Generating layout patterns with transformer via sequential pattern modeling," in *ICCAD*. IEEE, 2022.
[10] F. E. Gennari and Y.-C. Lai, "Topology design using squish patterns," Sep. 9 2014, uS Patent 8,832,621.
[11] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *NeurIPS*, vol. 33, pp. 6840–6851, 2020.
[12] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," in *ICLR*, 2020.
[13] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, Y. Shao, W. Zhang, B. Cui, and M.-H. Yang, "Diffusion models: A comprehensive survey of methods and applications," *arXiv preprint arXiv:2209.00796*, 2022.
[14] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Detecting multi-layer layout hotspots with adaptive squish patterns," in *DAC*, 2019, pp. 299–304.
[15] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. van den Berg, "Structured denoising diffusion models in discrete state-spaces," *NeurIPS*, vol. 34, pp. 17 981–17 993, 2021.
[16] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *MICCAI*. Springer, 2015, pp. 234–241.
[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *NeurIPS*, vol. 30, 2017.