DiffPattern-Flex: Efficient Layout Pattern Generation via Discrete Diffusion

Zixiao Wang, Wenqian Zhao, Yunheng Shen, Yang Bai, Guojin Chen, Farzan Farnia, Bei Yu

***Abstract*—Recent advancements in layout pattern generation have been dominated by deep generative models. However, relying solely on neural networks for legality guarantees raises concerns in many practical applications. In this paper, we present DiffPattern-Flex, a novel approach designed to generate reliable layout patterns efficiently. DiffPattern-Flex incorporates a new method for generating diverse topologies using a discrete diffusion model while maintaining a lossless and compute-efficient layout representation. To ensure legal pattern generation, we employ an optimization-based, white-box pattern assessment process based on specific design rules. Furthermore, fast sampling and efficient legalization technologies are employed to accelerate the generation process. Experimental results across various benchmarks demonstrate that DiffPattern-Flex significantly outperforms existing methods and excels at producing reliable layout patterns.**

***Index Terms*—Pattern Generation, Design For Manufacturability, Diffusion Models, Legalization**

I. Introduction

RELIABLE very-large-scale integration (VLSI) layout pattern libraries form the backbone of various Design for Manufacturability (DFM) research, such as refining design rules [1]–[3], optimizing Optical Proximity Correction (OPC) techniques [4]–[6], performing lithography simulations [7]–[9], and detecting layout hotspots [10]–[12]. With the increasing demand for layout patterns in machine-learning-based lithography design, building a comprehensive and practical large-scale pattern library has become highly resource-intensive due to the extended logic-to-chip design cycle.

To address this challenge, a variety of rule-based and learning-based layout pattern generation methods have been introduced. Early rule-based methods [13], [14] augmented predefined sets of basic units through simple techniques like flipping and rotation. These units were then randomly selected and combined. However, this approach results in limited diversity and quantity of generated patterns. More recently, learning-based generative methods [15]–[19] have demonstrated the ability to produce diverse layout patterns at a larger scale. Among these, pixel-based methods [15], [16] treat pattern generation as a binary image synthesis task. While layout patterns can be expansive, the distribution of critical information within them is often sparse. To mitigate computational inefficiency, a lossless pattern representation called *Squish Pattern* was introduced by [20], which compresses large layout patterns into smaller binary topology matrices and geometric vectors.

In pixel-based methods, a topology matrix with continuous values is synthesized and later thresholded to produce a binary matrix. This approach introduces inefficiencies and hampers model performance. Alternatively, sequential-based methods [17] model layout patterns as polygon sequences, which are decomposed into vertices and directed edges. These methods generate new layout patterns by creating polygon sequences that are then transformed into layouts. Both pixel-based and sequential approaches rely on the generative model to learn latent regularizations from the training data to prevent generating illegal patterns that violate design rules. However, we argue that these implicit constraints learned from the data are neither flexible nor reliable. When design rules change, these models often require retraining on large-scale datasets, and even then, a significant proportion of the generated patterns may violate the rules.

In this paper, we introduce DiffPattern-Flex, a practical and efficient pixel-based layout pattern generation framework composed of three key components:

Topology Generation: Inspired by the success of diffusion models [21], [22], we frame the topology generation task as a denoising problem. Using a discrete diffusion model, we predict the noise that should be removed at each step. Unlike prior work that applies thresholds on continuous outputs, our method uses discrete states (e.g., $\{0, 1\}$) to represent the image tensor, naturally aligning with the discrete nature of layout patterns. This discrete output reduces overfitting and avoids the need for manual thresholding.

Efficient Representation: Building on the *Squish Pattern* concept, we propose a more advanced lossless representation called *Deep Squish Pattern*. This method compresses the topology matrix into a topology tensor, significantly reducing input size while expanding the channel dimension. Since diffusion models are more sensitive to input size than to the number of input channels [23], this technique offers a computationally efficient solution applicable to other pixel-based generation methods.

Legalization Process: After generating topology matrices, we assign geometric vectors to them and restore legal layout

This work is supported by The Research Grants Council of Hong Kong SAR (No. CUHK14208021) and the MIND project (MINDXZ202404). (Corresponding author: Bei Yu)

Zixiao Wang, Wenqian Zhao, Yang Bai, Guojin Chen, Farzan Farnia and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong SAR.

Yunheng Shen is with Tsinghua University, Beijing, China.

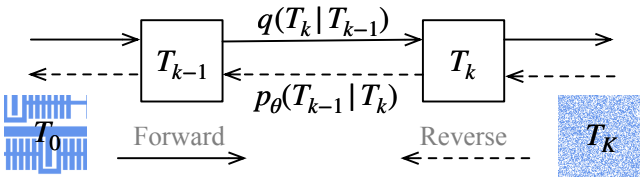


Fig. 1 Illustration of denoising diffusion process.

patterns. To achieve this, we designed a nonlinear system that provides a legal solution for each topology matrix and is adaptable to various design rules. Our white-box legalization strategy ensures a 100% legality rate for the generated patterns.

To further enhance diversity while maintaining pattern validity, we apply robust pattern augmentation techniques, with legality ensured via the white-box legalization process. Given the high demand for large-scale pattern libraries in downstream applications, we emphasize efficiency. To accelerate our framework, we introduce fast sampling techniques to speed up the discrete diffusion model's sampling process and employ data-guided initialization for optimal legalization.

Our main contributions are as follows:

1) We present a novel layout pattern generation method based on discrete denoising, capable of synthesizing layout topologies.
2) We propose a lossless layout pattern representation, *Deep Squish Pattern*, which improves the efficiency of pixel-based layout generation schemes.
3) We develop a white-box legalization system that ensures all generated patterns comply with design rules.
4) We integrate several optimization techniques that significantly speed up the generation process, achieving an $8.37\times$ improvement in sampling and a $2.48\times$ improvement in legalization.
5) We validate our approach through extensive experiments, demonstrating state-of-the-art (SOTA) performance on benchmark datasets.

The remainder of this paper is organized as follows. Section II introduces the prior knowledge and background. Section III provides detailed explanations of our framework. Section IV gives the details on how to enhance the diversity of generated patterns and how to accelerate both the sampling phase and the legalization phase. Section V presents the experimental results, followed by a conclusion in Section VI.

II. Preliminaries

A. Diffusion Models

Diffusion models, also known as denoising diffusion probabilistic models (DDPM) [21], [22], have shown great promise in generating high-quality images. These models employ a Markov chain [24] to describe both the forward and reverse diffusion processes, as depicted in Fig. 1. In the forward process, a sequence of noisy samples $\boldsymbol{T}_1, ..., \boldsymbol{T}_K$ is generated by progressively adding Gaussian noise to an original sample $\boldsymbol{T}_0$ over K steps. The noise level is governed 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 becomes large, the final noisy sample $\boldsymbol{T}_K$ approximates a Gaussian distribution, leading to the reverse diffusion process, which aims to generate new data samples from randomly drawn Gaussian noise. The reverse process aims to learn the inversion of the forward diffusion process, allowing the generation of fresh data. Since inverting this process requires an expressive model, we utilize a deep neural network with learnable parameters $\boldsymbol{\theta}$ to approximate the reverse distribution:

$$\boldsymbol{p_\theta}\left(\boldsymbol{T}_{k-1}|\boldsymbol{T}_k\right) := \mathcal{N}\left(\boldsymbol{T}_{k-1}; \boldsymbol{\mu_\theta}\left(\boldsymbol{T}_k, k\right), \boldsymbol{\Sigma_\theta}\left(\boldsymbol{T}_k, k\right)\right), \quad (2)$$

where, $\boldsymbol{\mu_\theta}$ and $\boldsymbol{\Sigma_\theta}$ represent the mean vector and covariance matrix of the distribution, respectively. The subscript $\boldsymbol{\theta}$ indicates that these quantities are obtained using a neural network with trainable parameters $\boldsymbol{\theta}$.

Similar to variational autoencoders (VAEs) [25], the training objective of diffusion models is to maximize the log-likelihood function by optimizing the variational lower bound (VLB):

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

where $L_k = D_{\mathrm{KL}}\left(\boldsymbol{q}\left(\boldsymbol{T}_{k-1}|\boldsymbol{T}_k, \boldsymbol{T}_0\right) \parallel \boldsymbol{p}_\theta\left(\boldsymbol{T}_{k-1}|\boldsymbol{T}_k\right)\right)$ and D_{KL} denotes the KL divergence. The term $\boldsymbol{q}\left(\boldsymbol{T}_{k-1}|\boldsymbol{T}_k, \boldsymbol{T}_0\right)$ can be computed as a Gaussian distribution derived from Equation (1) using Bayes' theorem.

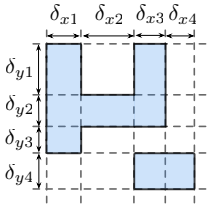
Once the diffusion model has been trained, new samples can be generated by sampling from a standard Gaussian distribution and iteratively removing noise using the reverse process, as described by Equation (2).

B. Squish Pattern Representation

A typical layout pattern consists of a collection of polygons, which often results in information sparsity, leading to unnecessary computational complexity and potential overfitting in neural network methods. The Squish Pattern representation [20] is an efficient, lossless encoding that compresses a layout into a topology matrix and two geometric vectors, Δ_x and Δ_y, as illustrated in Fig. 2. The layout is divided into grids based on scan lines that follow the polygon edges. The interval between adjacent scan lines is recorded in the Δ vectors. Each entry of the topology matrix is binary, where one indicates the presence of a shape, and zero denotes empty space. To maintain a consistent format, the squish pattern is padded into a square shape as outlined in [26].

C. Problem Formulation

A crucial metric for evaluating layout pattern generation is the diversity of the generated patterns. As defined in [15], 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 minus one along the



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.

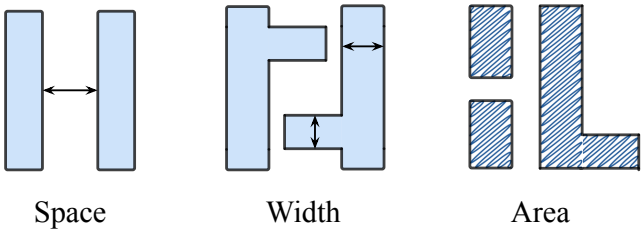


Fig. 3 Illustration of design rules.

x-axis and y-axis, respectively. Using this definition, we can express pattern diversity as follows:

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

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

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

Higher values of H indicate greater diversity within the pattern library, signifying a broader distribution of patterns.

In addition to diversity, layout patterns must adhere to certain design rules, as outlined in [16], [17]. As shown in Fig. 3, these design rules include 'Space' (the distance between adjacent polygons), 'Width' (the size of a shape in one direction), and 'Area' (the area of a polygon). Based on these metrics, we define:

Definition 2 (Pattern Legality)**.** *A layout pattern is considered legal if it is design rule check (DRC)-clean according to the specified design rules.*

With these evaluation metrics in mind, the pattern generation problem can be formally defined as:

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

III. Reliable Discrete Pattern Generation

A. Overview of DiffPattern-Flex

As illustrated in Fig. 4, our framework consists of three phases:

Deep Squish Pattern Encoding: For a given collection of layout patterns, we begin by deriving their deep squish pattern encoding. Each layout pattern is broken down into a topology tensor $\boldsymbol{T}$ along with two geometric vectors, $\boldsymbol{\Delta}_x$ and $\boldsymbol{\Delta}_y$.

Generation of Topology Tensors: The topology tensors $\boldsymbol{T}_0$ obtained from the extraction process are subsequently processed by a discrete diffusion model. In this model, noise is incrementally added to $\boldsymbol{T}$ with predetermined probabilities $q(\boldsymbol{T}_k|\boldsymbol{T}_{k-1})$, allowing the model to learn the inverse of this K-step noise application. To generate a new topology tensor $\hat{\boldsymbol{T}}$, a noise-infused tensor $\boldsymbol{T}_K$ is sampled, after which each component of $\boldsymbol{T}$ transitions among a set of finite states according to a predicted probability $p_\theta(\boldsymbol{T}_{k-1}|\boldsymbol{T}_k)$, eventually resulting in a plausible topology tensor $\hat{\boldsymbol{T}}_0$.

2D Legal Pattern Evaluation: For the generated topology tensors $\hat{\boldsymbol{T}}$, we employ an interpretable nonlinear system to allocate geometric vectors to each tensor, ensuring that the resulting layout complies with the specified *Design Rules*.

B. Deep Squish Pattern Representation

As discussed in Section II-B, the squish pattern provides a lossless representation of layout patterns, where the topology matrix is treated as a single-channel 2D binary mask, as depicted in Fig. 5 (left). However, the per-pixel information density remains suboptimal, given that the efficiency of diffusion models is more sensitive to image size than the number of pixel states. To address this issue, we propose a new representation method, the *Deep Squish Pattern*, which aims to provide a more compact encoding.

Consider the example shown in Fig. 5, where a topology matrix contains four adjacent pixels (2×2), each assigned a value of either zero or one, indicating shape or space. A straightforward way to increase information density would be to encode multiple pixel bits into one. However, concatenating bits from different pixels into a single state, such as assigning values from 0 to 15 to represent all possible states for a (2×2) pixel block, introduces unbalanced power to each bit. This imbalance can lead to numerical instability as the bit count increases. For instance, in a 4×4 pixel case, the first bit carries a power of 2^{15}, while the last bit holds a power of 1. Furthermore, the state space grows exponentially with the number of bits.

To resolve this, we observe that a state space of 16 (2^4) discrete states can be decomposed into permutations of four subspaces with two states each. Rather than assigning unequal powers to different bit positions, we assign equal weight to each bit by folding the squish topology matrix into a topology tensor $\boldsymbol{T}$ with multiple channels. In this process, a $\sqrt{C} \times \sqrt{C}$ patch from the topology matrix is transformed into a single point with C channels in the tensor $\boldsymbol{T}$. The hyper-parameter C is chosen to balance local information density and input size. This Deep Squish Pattern representation expands the model's effective receptive field and integrates naturally with pixel-based machine learning approaches. After the generation

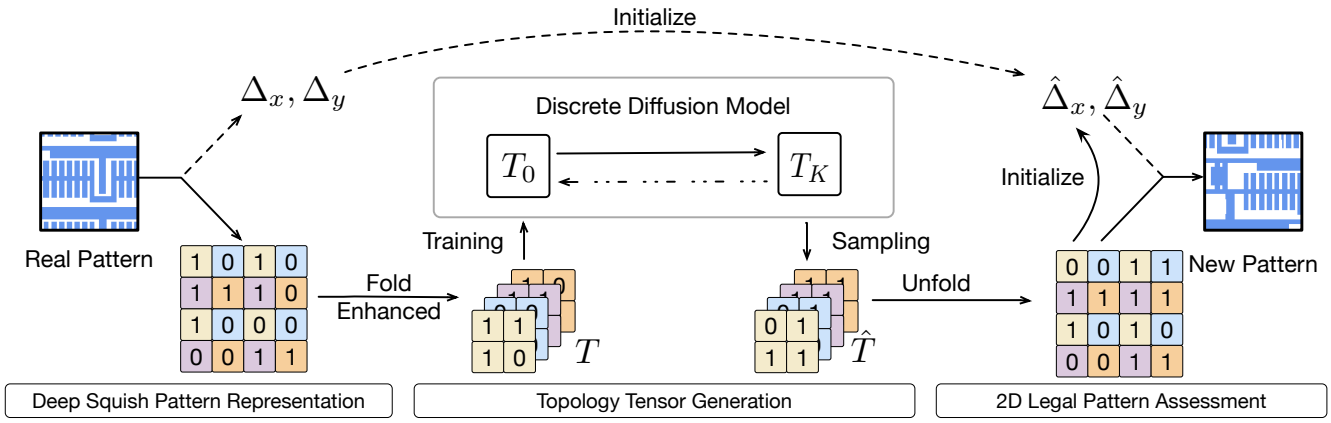


Fig. 4 An illustration of the Diffpattern-Flex framework for reliable layout pattern generation.

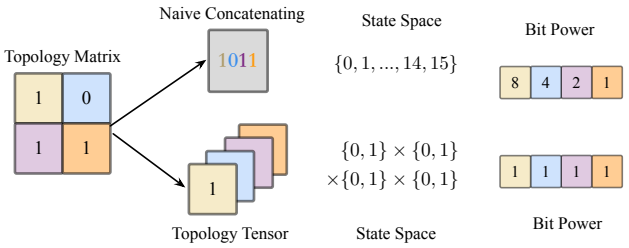


Fig. 5 An illustration of the Deep Squish Pattern Encoding. The Topology Tensor provides a compact and lossless representation of the topology matrix. Simple bit concatenation leads to unequal power distribution across bits and causes exponential growth in the state space.

process is completed, the original topology matrix can be restored by unfolding the topology tensor back into its matrix form. We should note that the folded topology tensor is still treated as a single sample rather than multiple separate samples during the generation process. Therefore, the deep squish pattern representation will not affect the logical resolution of the original topology matrix, preserving all information. When the topology generation is completed, the generated deep squish pattern will be flattened into a 2D topology matrix and further legalized in the subsequent steps.

C. Topology Tensor Generation

Once we have efficiently encoded the existing layout pattern using the Deep Squish Pattern representation, the next step is to learn the distribution of existing topology tensors and generate new ones with valid topology attributes. Let $\boldsymbol{T}_0 \in \{0,1\}^{C\times M\times M}$ represent a topology tensor extracted from existing patterns. A naive approach would treat the binary tensor as a grayscale image, learn the distribution through a diffusion model (as introduced in Section II-A), and convert the generated topology to a binary one by setting a threshold, as done in previous pixel-based pattern generation methods [15], [16]. However, forcing the network to learn discrete outputs (zero and one in our case) in a continuous state space from the training set is an inefficient use of the model's representational capacity. A more elegant approach is to generate discrete outputs naturally.

Discrete Diffusion Model. Unlike traditional diffusion models used in computer vision, we aim to synthesize the topology of layout patterns, where each entry in the topology belongs to a discrete state. We make several key modifications to enhance the diffusion model and directly synthesize discrete topology patterns. To achieve this, we first reformulate the problem.

At the k-th of K diffusion steps, $x_k \in \{0,1\}$ is an entry in the topology tensor $\boldsymbol{T}$. In the discrete diffusion model, a transition probability matrix $[\boldsymbol{Q}_k]_{ij} = q(x_k = j|x_{k-1} = i)$ describes 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 encoded version of the entry x_k, $\mathrm{Cat}(\boldsymbol{x}|\boldsymbol{p})$ represents a categorical distribution over the row vector $\boldsymbol{x}$ with probabilities given by the row vector $\boldsymbol{p}$, and $\boldsymbol{x}_{k-1}\boldsymbol{Q}_k$ is the row vector-matrix product. The matrix $\boldsymbol{Q}_k$ is applied independently to each entry in the topology tensor, and $\boldsymbol{q}$ factorizes over higher dimensions as well. The Deep Squish Pattern representation is well-suited for this discrete diffusion process, as the size of the transition matrix $\boldsymbol{Q}$ increases with the number of states for each pixel. Since every entry x in the topology tensor has only two states, we can efficiently model the transitions. During the reverse diffusion process, the neural network predicts the categorical distribution probability $\boldsymbol{p}_\theta(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k)$ for each entry to recover the original tensor.

The choice of transition probability matrix $\boldsymbol{Q}_k$ is critical, as it should ensure that the forward process $\boldsymbol{q}(\boldsymbol{x}_k|\boldsymbol{x}_0)$ converges to a known stationary distribution as k becomes large. A uniform stationary distribution is a natural choice for topology tensor generation, meaning that given any $\boldsymbol{x}_0$, the distribution of each entry $\boldsymbol{x}_k$ should follow:

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

Thus, we design a doubly stochastic matrix $\boldsymbol{Q}_k$ with strictly

positive entries for the 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 a hyperparameter controlling the noise level. To ensure that the model can accurately learn the original sample distribution and reach a stable distribution quickly, we adopt the classical setting from previous works [21], [27], using smaller noise in the early diffusion steps and larger noise in the later steps. 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 the Diffusion Model. To train the discrete diffusion model for topology tensor generation, the objective at each step k is to minimize the following loss function:

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

where λ is a hyperparameter balancing the loss terms.

Given a topology tensor $\boldsymbol{T}_0$, we randomly sample a target step k from 1 to K and generate a noisy sample $\boldsymbol{T}_k$. Fortunately, we can explicitly derive that $\boldsymbol{x}_k$ follows the 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 $\overline{\boldsymbol{Q}}_k = \boldsymbol{Q}_1 \boldsymbol{Q}_2 \ldots \boldsymbol{Q}_k$. Instead of adding noise at every step, we directly sample from this distribution to obtain $\boldsymbol{T}_k$.

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

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

where $\widetilde{\boldsymbol{x}}_0$ iterates over all possible states of $\boldsymbol{x}_0$. Using Bayes' theorem and Equation (5), we derive the closed form for $\boldsymbol{q}\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \boldsymbol{x}_0\right)$ as:

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

where $\odot$ is the element-wise (Hadamard) product.

At this point, all components of the loss function have been determined, allowing the diffusion model to be trained using gradient descent.

Generating Deep Squish Patterns. Once the training process is complete, we can synthesize new topology patterns by sampling a noise topology $\boldsymbol{T}_K$ from a uniform stationary distribution and iteratively removing the predicted noise using the reverse procedure. The sampling process is expressed as:

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

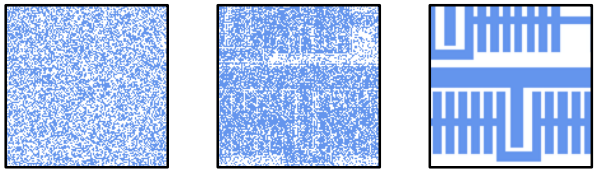


Fig. 6 An illustration of the (flattened) samples from our Discrete Diffusion Model.

where $\boldsymbol{T}_k$ represents the estimated pattern topology at step k, and $\hat{\boldsymbol{T}}_0$ is the newly generated topology tensor. The denoising procedure is illustrated in Fig. 6. The generated topology tensor $\hat{\boldsymbol{T}}_0$ is inherently binary, with each entry being either zero or one. After sampling is complete, we flatten $\hat{\boldsymbol{T}}_0$ for legal pattern assessment.

Topology Pre-filter. We apply a rule-based pre-filtering process to eliminate invalid topologies, such as Bow-tie shapes, based on domain knowledge. Thanks to the high quality of the topologies generated by our discrete diffusion model, less than 0.1% of the generated topologies are filtered out in our settings.

D. 2D Legal Pattern Assessment

Once the squish pattern generation is completed, the next step is to determine the legal Δ_x and Δ_y values for the generated topologies to create DRC-clean layout patterns. This decomposition of topology generation and legal pattern assessment gives DiffPattern-Flex the flexibility to adapt to changing design rules, as discussed in Section V-C. Instead of relying on black-box deep-learning methods as in previous works [16], [17], we employ a white-box approach to solve the problem. First, we list all constraints for each generated topology based on the design rules shown in Fig. 3, and then we formulate a nonlinear system that incorporates all these constraints, as represented in Equation (14):

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

where Space_{min} and Width_{min} are the lower bounds for 'Space' and 'Width'. $\sqrt{C}M \times \sqrt{C}M$ defines the dimensions of the topology matrix, and Area_{min} and Area_{max} define the permissible area range for each polygon in the pattern. All constants are pattern-independent and are provided by the design rules. Both Set_S and Set_W are pattern-dependent and indicate which pairs of scan lines are constrained by design rules on 'Space' and 'Width', respectively.

The nonlinear system in Equation (14) can be efficiently solved using nonlinear programming algorithms or numerical methods, and typically, there are multiple possible solutions.

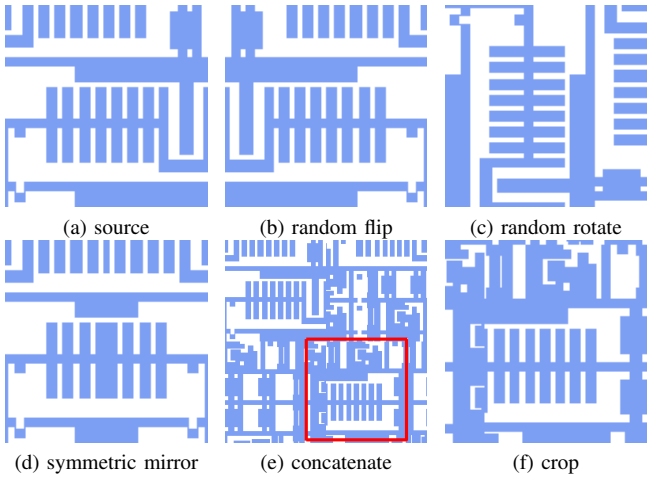


Fig. 7 Illustration of reliable pattern enhancement. In practice, multiple augmentation methods will be applied to the same pattern to enhance the data diversity. (f) denotes the random patch within the red box and is cropped from (e).

Each solution for Δ_x and Δ_y, together with the corresponding topology, constitutes a complete squish pattern representation. As discussed in Section V-C, DiffPattern-Flex can easily generate a large number of legal layout patterns from a single topology under given design rules. In rare cases, it may not be possible to find a legal solution in a limited time. Although this never occurred in our experiments (more than 1.0×10^8 attempts), we can simply discard these unsolvable cases from the generated topology set to avoid producing illegal patterns. In most cases, the choice of initial values for the nonlinear programming algorithms has minimal impact on pattern diversity and legality.

IV. Diversity and Acceleration

A. Reliable Topology Enhancement

Modern generative models are renowned for their scalability and the necessity of high-quality data. The proposed learning-based algorithm is expected to exhibit improved performance in pattern generation tasks when provided with more high-quality and diverse data. While extensive data augmentation techniques [28], [29] have been introduced in recent literature, directly applying methods from general image domains to the pattern domain is not advisable. Inappropriate pattern augmentation can compromise the validity of the augmented data and potentially degrade model performance.

To ensure the reliability of enhanced topology matrices, we propose a pre-checking process using the legalization method introduced in Section III-D. The reliable topology enhancement procedure is formulated as follows:

$$\widetilde{\mathfrak{T}} = \left\{ \widetilde{\boldsymbol{T}} = \Gamma(\boldsymbol{T}) \;\middle|\; \Xi(\widetilde{\boldsymbol{T}}) = \text{Success} \right\}, \tag{15}$$

where $\boldsymbol{T}$ and $\widetilde{\boldsymbol{T}}$ represent the topology matrices before and after enhancement, respectively. The enhanced dataset $\widetilde{\mathfrak{T}}$ is the collection of $\widetilde{\boldsymbol{T}}$. Here, $\Gamma(\cdot)$ denotes a data augmentation function, which consists of various existing enhancement techniques, and $\Xi(\cdot)$ refers to the legalization method proposed in Section III-D. Although various augmentation methods may be employed, only topologies successfully legalized by the function $\Xi(\cdot)$ are used during the training of diffusion models.

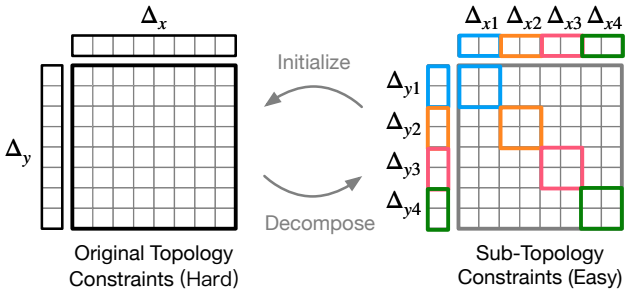


Fig. 8 Illustration of legalization acceleration. Each diagonal block denotes an independent sub-problem, and all sub-problems can be solved in parallel.

In our implementation, the data augmentation function $\Gamma(\cdot)$ consists of (1) *Random Flip*, (2) *Random Rotation*, (3) *Symmetric Mirror*, and (4) *Concatenate and Crop*. Several examples of these augmentation techniques are provided in Fig. 7. These methods were chosen because they do not introduce excessive perturbations, such as meaningless random noise, into the topology matrices. Overly noisy data is rare in practical scenarios and can negatively impact downstream tasks [30], [31].

Pre-checking with the legalization function $\Xi(\cdot)$ is crucial due to the inevitable noise introduced by augmentation methods. A potential concern is the increased time required for optimizing Equation (14), which scales with the size of the topology matrix $\boldsymbol{T}$. This issue will be further discussed in the next subsection.

B. Acceleration

Given the large demands of pattern libraries, the efficiency of the pattern generation framework is crucial. In DiffPattern-Flex, both the topology-generation phase and the legalization phase can be further accelerated as detailed below.

Fast-Sampling. When generating new topologies from well-trained models, a straightforward approach is to gradually reduce the noise in the topology $\boldsymbol{T_k}$ and reduce the step k by one during inference, as explained in Equation (11) and Equation (13). However, considering the large step number used in practice ($K = 1000$ in our implementation), the generation process can be time-consuming. To accelerate the inference procedure, we observe that it is possible to perform inference with m steps at a time by modifying Equation (11) into

$$\boldsymbol{p_\theta}\left(\boldsymbol{x}_{k-m}|\boldsymbol{x}_k\right) = \sum_{\widetilde{\boldsymbol{x}}_0} \boldsymbol{q}\left(\boldsymbol{x}_{k-m}|\boldsymbol{x}_k, \widetilde{\boldsymbol{x}}_0\right) \boldsymbol{p_\theta}\left(\widetilde{\boldsymbol{x}}_0|\boldsymbol{x}_k\right), \tag{16}$$

where $\boldsymbol{q}\left(\boldsymbol{x}_{k-m}|\boldsymbol{x}_k, \widetilde{\boldsymbol{x}}_0\right)$ has a closed form by extending Equation (12) into

$$\boldsymbol{q}\left(\boldsymbol{x}_{k-m}|\boldsymbol{x}_k, \boldsymbol{x}_0\right) = \mathrm{Cat}\left(\boldsymbol{x}_{k-m}; \boldsymbol{p} = \frac{\boldsymbol{x}_k \boldsymbol{Q}_k^\top \odot \boldsymbol{x}_0 \overline{\boldsymbol{Q}}_{k-m}}{\boldsymbol{x}_0 \overline{\boldsymbol{Q}}_k \boldsymbol{x}_k^\top}\right). \tag{17}$$

By performing inference with m steps at a time, the topology generation can be roughly accelerated by a factor of m. However, the choice of m is a trade-off between the quality of topologies and speed. A higher m accelerates inference at the cost of reduced quality, while a lower m provides better quality at the expense of slower inference.

Efficient Implementation of Legalization. The time required for the legalization phase increases significantly with the size of the topology matrix [32]. Several works [2], [33], [34] have explored acceleration methods for this process in optimization literature. Considering the special form of constraints in our case, some customized techniques can be applied to accelerate the optimization process.

Empirically, we found that a good initialization improves the convergence of the optimization process. In practice, when the generated patterns follow the same design rules as the source patterns, we randomly select a pair of existing geometric vectors from the dataset as the starting point. This empirically accelerates the convergence of the nonlinear programming algorithm. A potential explanation is that existing geometric vectors are more likely to partially satisfy the constraints of the optimization problem. However, when design rules change or suitable geometric vector pairs are unavailable, a general initialization method becomes critical.

To address this issue, we extend the idea of 'partially satisfying constraints' and initialize the optimization problem using solutions from several independent sub-problems, as illustrated in Fig. 8. Specifically, we solve several independent sub-problems, each constrained by one diagonal block in the original topology matrix. The solving processes of sub-problems are independent and can be efficiently optimized in a parallel fashion. The solution of each sub-problem satisfies the constraints within its block and is concatenated with others to initialize the original optimization problem, thus accelerating the process.

V. Experimental Results

A. Experimental Setup

Datasets. In line with prior research [16], [17], we utilize a dataset comprising small layout pattern images sized 2048×2048 nm^2, derived by segmenting a 400×160 μm^2 layout map from the ICCAD 2014 contest. The extracted topology tensor is maintained at a fixed size of $16\times32\times32$ with $C = 16$, as used in the Deep Squish Pattern Representation framework.

Diffusion Model Configuration. In line with previous research [21], [27], we adopt a U-Net architecture [35] as the foundation of our discrete diffusion model to approximate the posterior distribution during the reverse diffusion process. The model operates at four different feature map resolutions: $[32 \times 32, 16 \times 16, 8 \times 8, 4 \times 4]$. Each resolution stage consists of two convolutional residual blocks, with the number of convolutional channels set to $[128, 256, 256, 256]$, respectively. At the 16×16 resolution stage, a self-attention block is inserted between the two convolutional blocks. Moreover, the time step k is embedded into each residual block using sinusoidal positional encodings [36]. To guarantee convergence of the forward diffusion process to a uniform stationary distribution, we set the number of diffusion steps K to 1000. The noise schedule β_k increases linearly from 0.01 to 0.5. During inference, we found that reversing the process in $m = 10$ steps offers an effective trade-off between pattern quality and computational efficiency, as discussed in Section IV-B.

Training Details. The diffusion model is trained for 0.5M iterations with a batch size of 128, using a learning rate of $2e$-4 and the Adam optimizer. The following hyperparameters are used: a dropout rate of 0.1, gradient clipping set to 1, and a loss coefficient λ of 0.001. As described in Section IV-A, data augmentation is applied randomly during training using reliable augmentation techniques. The probabilities for the augmentation methods—*Random Flip*, *Random Rotation*, *Symmetric Mirror*, and *Concatenate and Crop*—are $[0.5, 1.0, 0.5, 0.5]$. The training process spans approximately 20 hours on 8 NVIDIA RTX 3090 GPUs.

B. Pattern Diversity and Legality

To evaluate the models [37], [38], we calculate the diversity of the generated patterns using Equation (4), while the legality of the patterns is verified with the tool *Klayout*, which checks compliance with the design rules as outlined in Section II-C.

For a fair comparison with previous methods, we randomly synthesize 100,000 topologies, assigning each generated topology a pair of geometric vectors during the legality assessment phase. In this evaluation, we refer to our method as DiffPattern-Flex. It is worth noting that DiffPattern-Flex can generate a large number of legal patterns for each topology, as the solution to the optimization problem in Equation (14) is not unique.

We compare DiffPattern-Flex with several learning-based layout pattern generation approaches. Specifically, CAE [15] represents a standard convolutional autoencoder model, while VCAE [16] utilizes a variational convolutional autoencoder. Both of these methods are pixel-based. LegalGAN [16] is a post-processing approach that legalizes generated topologies by making necessary modifications. In contrast, LayouTransformer [17] employs a sequence-based transformer model to synthesize new sequential representations of layout patterns, bypassing direct topology generation. Lastly, DiffPattern [18], our primary competitor, uses a discrete diffusion model to generate topology matrices. We also implemented the rule-based pattern generation method in [14]. An original pattern is split into four sub-patterns. The sub-patterns are further randomly flipped and rotated and form an enhanced sub-pattern library. A new pattern consists of four randomly

TABLE I Numerical results on pattern diversity and legality. 'Real Patterns' are from the ICCAD 2014 contest. '-' denotes an inapplicable result. Results of learning-based baseline are from previous works [18].

Set/Method	Generated Topology	Generated Patterns		Legal Patterns	
		Patterns	Diversity (↑)	Legality (↑)	Diversity (↑)
Real Patterns	-	-	-	13869	10.777
CAE [15]	100000	100000	4.5875	19	3.7871
VCAE [16]	100000	100000	10.9311	2126	9.9775
CAE+LegalGAN [16]	100000	100000	5.8465	3740	5.8142
Rule-based [14]	-	100000	10.256	24474	10.066
VCAE+LegalGAN [16]	100000	100000	9.8692	84510	9.8669
LayouTransformer [17]	-	100000	10.532	89726	10.527
DiffPattern [18]	100000	100000	10.815	**100000**	10.815
Ours	100000	100000	**11.713**	**100000**	**11.713**

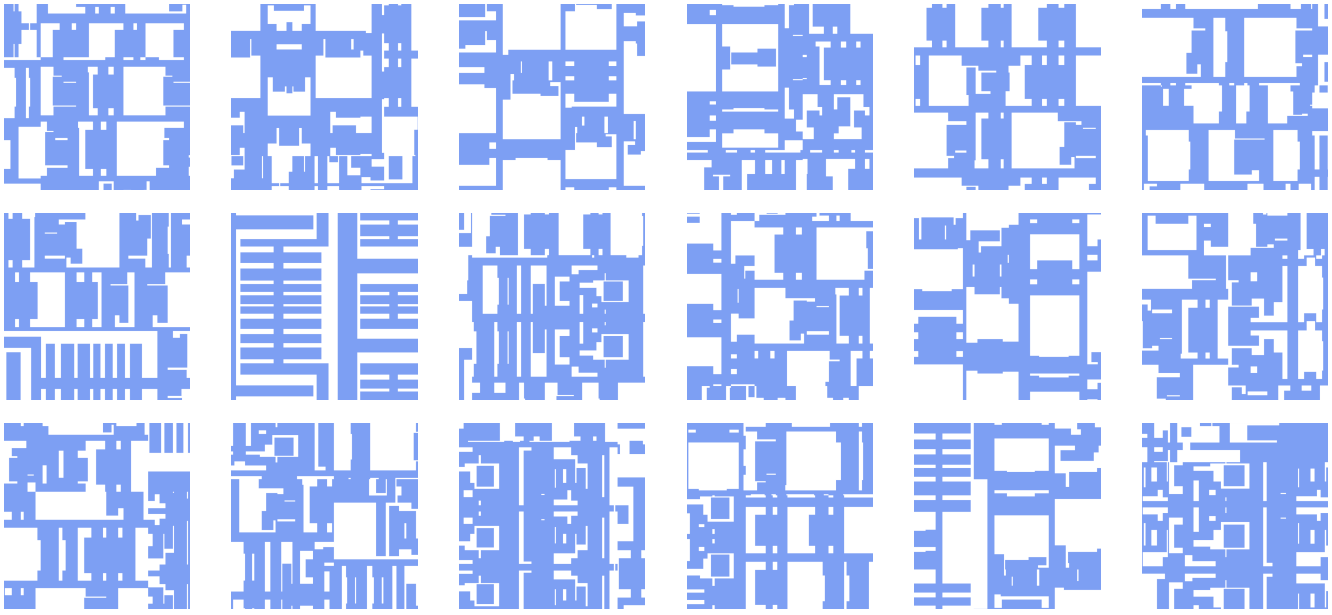


Fig. 9 Some randomly selected examples generated by our method.

picked sub-patterns from the library. We also generate 100,000 patterns in this way and test the diversity and legality of the generated patterns.

As presented in TABLE I, DiffPattern-Flex outperforms other methods in terms of legality, achieving a perfect performance (*i.e.*, 100%) under standard conditions. This is due to the topology pre-filtering and rule-based 2D legal pattern assessment. Furthermore, DiffPattern-Flex demonstrates a marked improvement in diversity (from 10.815 to 11.713) over DiffPattern, thanks to its pattern augmentation techniques. The robustness of the generation and assessment methods ensures the reasonableness of the augmented patterns.

Illustrative examples generated by DiffPattern-Flex are shown in Fig. 9, highlighting the diversity of the patterns produced. These examples underscore the capability of our method to generate detailed, reasonable patterns that maintain a high degree of diversity across the dataset.

Moreover, the legality of patterns produced by DiffPattern-Flex is consistently ensured through its rule-based legal pattern assessment and topology pre-filtering process. When design rules are modified, our method allows users to quickly generate a new batch of diverse, compliant patterns without retraining the topology generation model. This flexibility is explored further in the next subsection.

C. Flexibility

A key advantage of DiffPattern-Flex is the flexibility provided by its white-box 2D legal pattern assessment phase. Below, we illustrate two applications that leverage this flexibility.

Generating Multiple Patterns from a Single Topology. Given a set of design rules and a specific topology, the non-linear system in Equation (14) often admits numerous legal solutions, *i.e.,* different geometric vector pairs. Each legal solution corresponds to a valid layout pattern, and these patterns, while differing in geometric vectors, share the

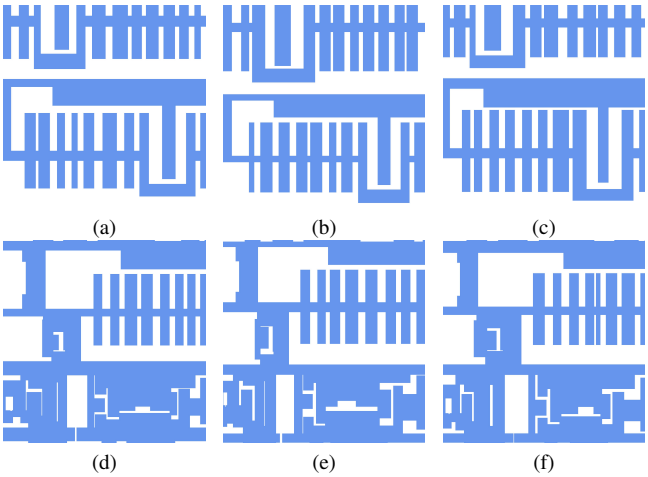


Fig. 10 Different layout patterns that are generated from a single topology with the same design rule.

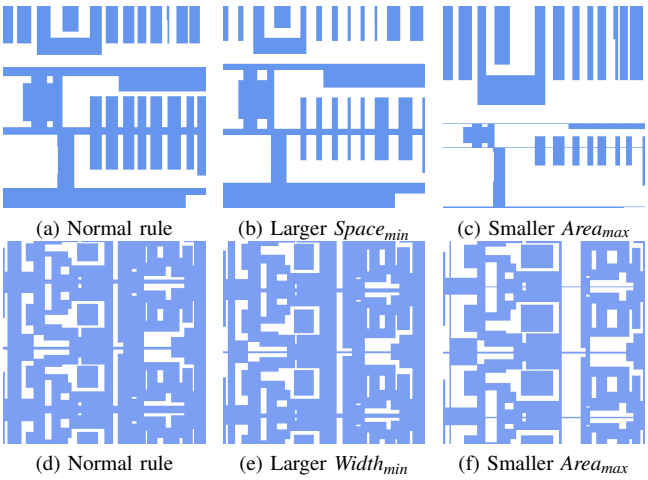


Fig. 11 Layout patterns that are generated from the same topology with different design rules.

same underlying topology. Such variations can be valuable for certain downstream tasks. Fig. 10 demonstrates examples where multiple layout patterns are generated from a single topology by using different geometric vectors.

Generating Legal Patterns under Varying Design Rules The decoupling of legalization and topology generation in DiffPattern-Flex enables the flexibility of design rule modification. Since changes in design rules do not affect the distribution of topology matrices, the trained topology generation model remains applicable across varying design rules. Fig. 11 presents examples where multiple layout patterns are generated from a single topology, each adhering to a distinct set of design rules.

In this paper, most cases can be resolved within a reasonable time budget using the design rules defined in Section II-C. The experimental results demonstrate that the proposed legalization method can handle changes in the constants of the design rules. However, it is important to note that in scenarios where the design rules are extremely strict and complex, the generated topology matrices may fail to find a legal solution within a limited time using the proposed legalization methods. In such cases, advanced legalization techniques should be developed to reduce computational costs, which we leave for future work.

D. Distribution of Complexity

Diversity is a vital metric for assessing the quality of the generated pattern library. As outlined in Section II-C, diversity is quantified using the Shannon entropy of the distribution of pattern complexity, *i.e.*, the number of scan lines that intersect a pattern along both the x-axis and y-axis. The distribution of complexity is visualized in Fig. 12.

The patterns generated by DiffPattern exhibit a complexity distribution comparable to that of real-world patterns. However, due to the robust pattern augmentation techniques, the patterns generated by DiffPattern-Flex show increased diversity. The heatmap generated by DiffPattern-Flex covers regions that are missing from the heatmap of real patterns, suggesting that DiffPattern-Flex is capable of discovering novel combinations of existing patterns.

This visualization underscores our capability to produce high-quality, diverse layout patterns.

E. Model Efficiency

Efficiency is a critical measure when evaluating methods for layout generation. In our approach, since the processes of topology generation and layout pattern validation are decoupled, we separately record the average time taken to sample a new topology and to solve for a legal solution of Equation (14). The corresponding results are provided in Tables II and III.

As outlined in Section IV-B, the sampling procedure can be accelerated by approximately a factor of m, where m represents the number of backward steps executed during each model inference. In our implementation, we set $m = 10$, and the findings show that the sampling process can be sped up by $8.37\times$ with only a slight reduction in the diversity of the generated patterns.

In scenarios where both the target and source patterns adhere to the same design rules, as mentioned in Section III-D, an existing pair of geometric vectors can be randomly chosen to initialize the nonlinear system, which significantly speeds up the process. This method is referred to as *Solving-E*. When compared to the version using random initialization, termed *Solving-R*, *Solving-E* offers an average acceleration of $2.30\times$. Several examples illustrating the outcomes of *Solving-E* are shown in Fig. 13.

In scenarios where the target pattern and source pattern follow different design rules or when suitable geometric vectors are unavailable, we can accelerate the solving process by providing a better initialization using the divide method described in Section IV-B. This version is denoted as *Solving-D*. Our experimental results show that *Solving-D* achieves an average acceleration of $2.48\times$ in our cases. The total time for *Solving-D* includes both the solution of sub-problems and

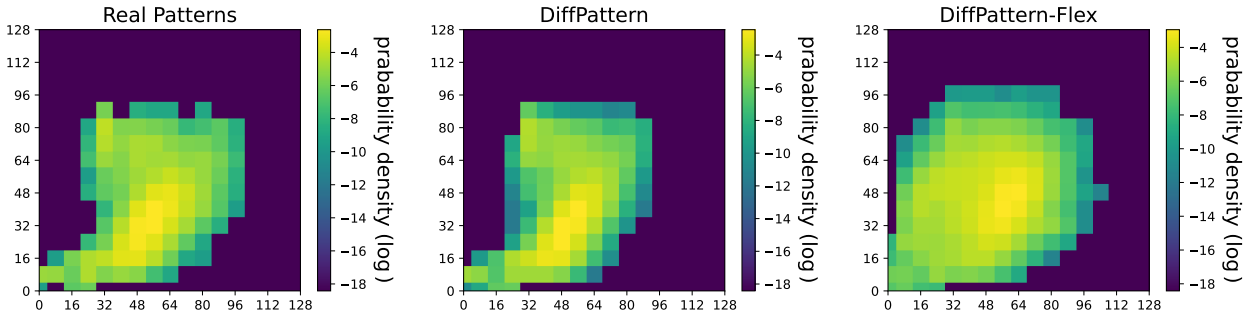


Fig. 12 An illustration of complexity distribution. Reliable pattern augmentation enriches the pattern distribution while maintaining the reasonability of the enhanced patterns.

TABLE II Model efficiency of DiffPattern-Flex under different accelerator factor m. Nvidia RTX 3090 GPU is used for topology sampling in this table.

Method	m	Cost Time (s)	Acceleration	Diversity
Sampling	1	0.544	1.00×	11.724
Fast-Sampling	5	0.118	4.61×	11.715
Fast-Sampling	10	0.065	8.37×	11.713
Fast-Sampling	20	0.039	13.95×	10.573

TABLE III Model efficiency of DiffPattern-Flex. Intel(R) Xeon(R) Gold 6326 CPU @ 2.90GHz is used to figure out the nonlinear system in this table.

Method	Extra Information	Cost Time (s)	Acceleration
Solving-R	Not required	0.269	1.00×
Solving-E	Required	0.117	2.30×
Solving-D	Not required	0.108	2.48×

the main problem. The divide size in our implementation is fixed as 4, which means we divide the hard main problem into four sub-problems for each tensor matrix. We also show some cases in Fig. 13. In the major cases, the final solution has a mirror difference compared with the initialization obtained from sub-problems.

F. Ablation Study

In this subsection, we conduct some ablation studies on how each part in DiffPattern-Flex affects the results.

Comparison with Continual Diffusion Model. In the main part of this paper, we utilize a discrete diffusion model to directly generate a discrete topology matrix, fully leveraging the capacity of the neural network. Here, we compare our method with a continuous diffusion baseline [21] to demonstrate the effectiveness of discrete modeling. For a fair comparison, we adopted the same training protocols as in our paper, including the dataset, augmentation methods, batch size, learning rate, optimization methods, and other hyperparameters. We also employed a U-Net [35] with a similar architecture to ensure comparable model capacity to our method. The primary difference is that the continuous diffusion model predicts a continuous tensor at each iteration, and the output of the final iteration is binarized using a fixed threshold of 0.5. After training, we evaluated the performance. Since the legality of the generated patterns is guaranteed by the proposed deep squish tensor and legalization method, we focused on comparing the diversity of the generated patterns. As shown in TABLE IV, discrete modeling improves diversity by a reasonable margin (11.294 → 11.713). The results support our claim in the main text and indicate that discrete modeling of the topology tensor benefits the layout pattern generation task.

TABLE IV Comparison between discrete modeling and continual modeling in topology tensor generation.

Modeling	Generated Topology	Diversity
Continual	100000	11.294
Discrete	100000	**11.713**

Probability of Concatenate and Crop Augmentation We conduct an ablation study to investigate the impact of the probability of concatenate and crop augmentation on the final results. By gradually increasing the augmentation probability from 0.0 to 1.0 during training, we retrain the model on the augmented data and evaluate the diversity of the generated samples. The results, presented in Fig. 14, indicate that the

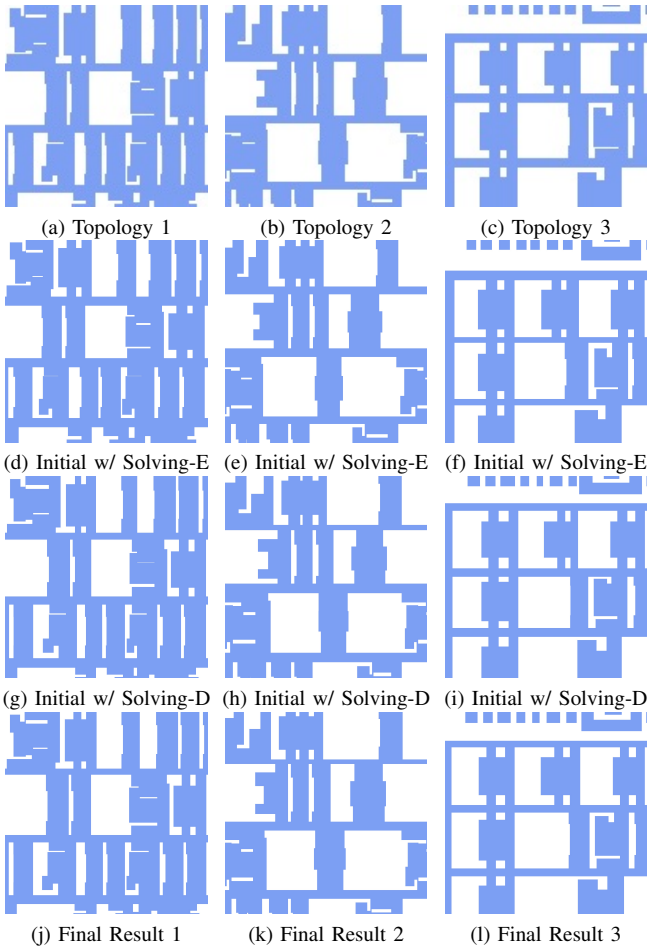


Fig. 13 Illustration of initialization and final solution in the legalization phase.

optimal probability range lies within $[0.5, 0.9]$. Based on these findings, we set the probability to 0.5 in our implementation.

Fast-Sampling Factor m. In our method, the fast-sampling factor m can be adjusted to balance efficiency and model performance. We extend the discussion in the main text by testing various values of m. The results are presented in TABLE II. We observe that $m = 10$ provides a good trade-off between efficiency and model performance. For cases where $m > 10$, the diversity of generated samples decreases significantly.

G. Discussion on Validity

A metric referred to as pattern validity was introduced in previous work [16]. This metric is evaluated using an encoder-decoder model pre-trained on the training data. The underlying assumption is that generated patterns that resemble those in the training set will achieve higher scores in this validity metric. However, we contend that this interpretation of validity is somewhat narrow. One of the key objectives in layout pattern generation is to produce a diverse array of legal patterns for various downstream tasks, such as hotspot detection or lithography simulation. In these cases, legal patterns that diverge from the training set are often more desirable, yet they tend to receive lower scores under the current validity metric.

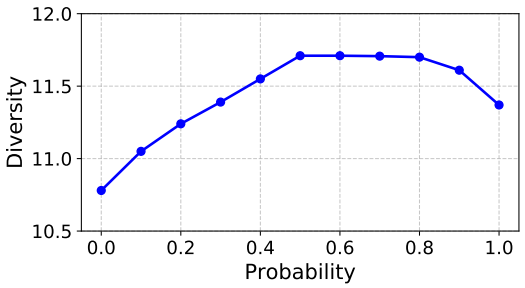


Fig. 14 The effect of different probabilities of concatenate and crop augmentation.

What further undermines the usefulness of this metric is that it promotes overfitting to the training set. For example, as noted in [16], [17], the generated patterns can obtain significantly higher validity scores (from 65% to 84%) compared to patterns in the test set, which should share the same distribution as the training data. It is unrealistic to assume that a higher validity score necessarily correlates with better-quality patterns.

For these reasons, we have opted not to assess DiffPattern-Flex using this metric.

VI. Conclusion

In this paper, we proposed DiffPattern-Flex, a novel framework for efficient and legal layout pattern generation. By decoupling topology generation and pattern legalization, our method provides flexibility in handling changing design rules without retraining the model, making it highly adaptable for various downstream tasks in design automation. Our approach demonstrates significant improvements in both the diversity and legality of generated patterns, achieving state-of-the-art performance. The ability to generate diverse legal patterns enhances the robustness of machine-learning workflows and supports a wide range of applications, such as lithography simulation and hotspot detection. Future work will focus on expanding DiffPattern-Flex to more complex tasks, such as multi-source pattern generation and integration of additional design constraints.

References

[1] Z. He, Y. Ma, and B. Yu, "X-check: Gpu-accelerated design rule checking via parallel sweepline algorithms," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2022, pp. 1–9.

[2] H. Modarres and R. J. Lomax, "A formal approach to design-rule checking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 6, no. 4, pp. 561–573, 1987.

[3] J. Jiang, L. Zou, W. Zhao, Z. He, T. Chen, and B. Yu, "Pdrc: Package design rule checking via gpu-accelerated geometric intersection algorithms for non-manhattan geometry," in *ACM/IEEE Design Automation Conference (DAC)*, 2024.

[4] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "MOSAIC: Mask optimizing solution with process window aware inverse correction," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[5] O. W. Otto, J. G. Garofalo, K. Low, C.-M. Yuan, R. C. Henderson, C. Pierrat, R. L. Kostelak, S. Vaidya, and P. Vasudev, "Automated optical proximity correction: a rules-based approach," in *Optical/Laser Microlithography VII*, vol. 2197. SPIE, 1994, pp. 278–293.

[6] S. Zheng, G. Xiao, G. Yan, M. Dong, Y. Li, H. Chen, Y. Ma, B. Yu, and M. Wong, "Model-based opc extension in openilt," in *International Symposium of Electronics Design Automation (ISEDA)*. IEEE, 2024, pp. 568–573.

[7] J. Kuang and E. F. Young, "An efficient layout decomposition approach for triple patterning lithography," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2013, pp. 1–6.

[8] B. Yu, K. Yuan, D. Ding, and D. Z. Pan, "Layout decomposition for triple patterning lithography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 34, no. 3, pp. 433–446, 2015.

[9] G. Chen, Z. Wang, B. Yu, D. Z. Pan, and M. D. Wong, "Ultra-fast source mask optimization via conditional discrete diffusion," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2024.

[10] R. Chen, W. Zhong, H. Yang, H. Geng, X. Zeng, and B. Yu, "Faster region-based hotspot detection," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[11] H. Yang, J. Su, Y. Zou, B. Yu, and E. F. Young, "Layout hotspot detection with feature tensor generation and deep biased learning," in *ACM/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[12] H. Geng, H. Yang, L. Zhang, F. Yang, X. Zeng, and B. Yu, "Hotspot detection via attention-based deep layout metric learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 8, pp. 2685–2698, 2022.

[13] G. R. Reddy, C. Xanthopoulos, and Y. Makris, "Enhanced hotspot detection through synthetic pattern generation and design of experiments," in *IEEE VLSI Test Symposium (VTS)*. IEEE, 2018, pp. 1–6.

[14] W. Ye, Y. Lin, M. Li, Q. Liu, and D. Z. Pan, "LithoROC: lithography hotspot detection with explicit ROC optimization," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 292–298.

[15] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "DeePattern: Layout pattern generation with transforming convolutional auto-encoder," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[16] 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)*. IEEE, 2020, pp. 1–9.

[17] 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 *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2022.

[18] Z. Wang, Y. Shen, W. Zhao, Y. Bai, G. Chen, F. Farnia, and B. Yu, "DiffPattern: Layout pattern generation via discrete diffusion," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[19] Z. Wang, Y. Shen, X. Yao, W. Zhao, Y. Bai, F. Farnia, and B. Yu, "ChatPattern: Layout Pattern Customization via Natural Language," in *ACM/IEEE Design Automation Conference (DAC)*, 2024.

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

[21] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 6840–6851, 2020.

[22] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," in *International Conference on Learning Representations (ICLR)*, 2020.

[23] 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.

[24] J. R. Norris, *Markov chains*. Cambridge university press, 1998, no. 2.

[25] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[26] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Detecting multi-layer layout hotspots with adaptive squish patterns," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 299–304.

[27] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. van den Berg, "Structured denoising diffusion models in discrete state-spaces," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 17 981–17 993, 2021.

[28] D. A. Van Dyk and X.-L. Meng, "The art of data augmentation," *Journal of Computational and Graphical Statistics*, vol. 10, no. 1, pp. 1–50, 2001.

[29] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.

[30] N. Natarajan, I. S. Dhillon, P. K. Ravikumar, and A. Tewari, "Learning with noisy labels," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 26, 2013.

[31] Z. Wang, J. Weng, C. Yuan, and J. Wang, "Truncate-split-contrast: a framework for learning from mislabeled videos," in *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 37, no. 3, 2023, pp. 2751–2758.

[32] D. Kraft, "A software package for sequential quadratic programming," *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[33] C. T. Lawrence and A. L. Tits, "A computationally efficient feasible sequential quadratic programming algorithm," *Siam Journal on optimization*, vol. 11, no. 4, pp. 1092–1118, 2001.

[34] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta numerica*, vol. 4, pp. 1–51, 1995.

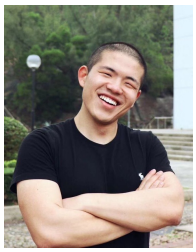
[35] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer, 2015, pp. 234–241.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[37] L. Theis, A. v. d. Oord, and M. Bethge, "A note on the evaluation of generative models," *arXiv preprint arXiv:1511.01844*, 2015.

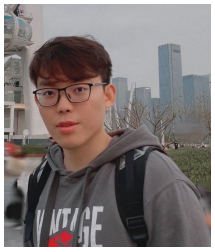
[38] Z. Wang, F. Farnia, Z. Lin, Y. Shen, and B. Yu, "On the evaluation of generative models in distributed learning tasks," *arXiv preprint arXiv:2310.11714*, 2023.

Zixiao Wang received his B.Eng. degree in Automation from Tsinghua University (THU) in 2019 and an MSc.degree in Computer Science from Tsinghua University (THU) in 2022. He is currently pursuing his Ph.D. degree at the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include Generative AI empowers EDA.



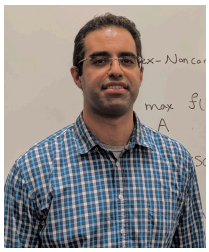
Wenqian Zhao obtained his Ph.D. and B.Sc. of Computer Science and Engineering from The Chinese University of Hong Kong, Hong Kong, in 2024 and 2019. His research interests include machine learning for VLSI design automation and hardware-aware deep- learning acceleration.

Yunheng Shen received his B.Eng. degree in Automation from Tsinghua University (THU) in 2019 and is currently pursuing his Ph.D. degree at the Department of Automation, THU. His current research interests include federated learning and the application of generative models or large models at the edge.



Yang Bai received the BS degree in telecommunications engineering from Xidian University in 2017, and Master degree in computer science from the Chinese Academy of Sciences University in 2020. He is a Ph.D. in Department of Computer Science and Engineering, the Chinese University of Hong Kong. His research interests focus on the optimization for deep neural network training and inference via compilation techniques.

Guojin Chen received his B.Eng. degree in software engineering from Huazhong University of Science and Technology (HUST) in 2019. He is currently pursuing his Ph.D. degree at the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His current research interests include (1) machine learning in VLSI design for manufacturability and (2) physics-informed networks for solving EDA area problems.



Farzan Farnia is an Assistant Professor of Computer Science and Engineering at The Chinese University of Hong Kong. Prior to joining CUHK, he was a postdoctoral research associate at the Laboratory for Information & Decision Systems, Massachusetts Institute of Technology, from 2019-2021. He received his M.Sc. and Ph.D. degrees in Electrical Engineering from Stanford University where he was a graduate research assistant at the Information Systems Laboratory advised by David Tse. He also received his B.Sc. degree in Electrical Engineering and Mathematics from Sharif University of Technology. His research interests lie in learning and information sciences with a particular focus on multi-learner learning frameworks where he study the convergence, equilibrium, and robustness properties of multi-learner learning algorithms.

Bei Yu (M'15-SM'22) received the Ph.D. degree from The University of Texas at Austin in 2014. He is currently an Associate Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He has served as TPC Chair of ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees. He received eleven Best Paper Awards from ICCAD 2024 & 2021 & 2013, IEEE TSM 2022, DATE 2022, ASPDAC 2021 & 2012, ICTAI 2019, Integration, the VLSI Journal in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, six ICCAD/ISPD contest awards, IEEE CEDA Ernest S. Kuh Early Career Award in 2021, DAC Under-40 Innovator Award in 2024, and Hong Kong RGC Research Fellowship Scheme (RFS) Award in 2024.