ChatPattern: Layout Pattern Customization via Natural Language

Zixiao Wang* CUHK
Yunheng Shen* Tsinghua University
Xufeng Yao CUHK
Wenqian Zhao CUHK
Yang Bai CUHK
Farzan Farnia CUHK
Bei Yu CUHK

Abstract

Existing works focus on fixed-size layout pattern generation, while the more practical free-size pattern generation receives limited attention. In this paper, we propose ChatPattern, a novel Large-Language-Model (LLM) powered framework for flexible pattern customization. ChatPattern utilizes a two-part system featuring an expert LLM agent and a highly controllable layout pattern generator. The LLM agent can interpret natural language requirements and operate design tools to meet specified needs, while the generator excels in conditional layout generation, pattern modification, and memory-friendly patterns extension. Experiments on challenging pattern generation setting shows the ability of ChatPattern to synthesize high-quality large-scale patterns.

1 Introduction

High-quality Very-Large-Scale Integration (VLSI) layout pattern libraries are foundational to numerous Design for Manufacturability (DFM) studies, such as refining design rules, formulating Optical Proximity Correction (OPC) recipes [1], conducting lithography simulations [2, 3], and detecting layout hotspots [4]. Contemporary machine-learning-based lithography design applications typically require an extensive array of layout patterns to train their networks, yet assembling a large-scale pattern library is cost-prohibitive due to the intricate logic-to-chip design cycle.

Prior to the advent of machine learning, rule-based methods [5] were employed to synthesize layout patterns automatically, with simple hand-drafted augmentations like flipping and rotation used to expand the fundamental layout pattern units. These units would then be assembled randomly to create larger designs. However, the diversity and volume of patterns generated by these rule-based methods often fell short of user expectations. Conversely, recent learning-based methods [6–9] have demonstrated the ability to generate a plethora of diverse layout patterns that closely match the dataset distribution. One notable technique, the squish pattern trick [10], has been instrumental in reducing computational and memory demands by condensing a layout pattern patch into a compact 2D-topology matrix, upon which generative models are trained. Once the topology matrices are generated, legalization technologies are employed to recover the topology matrices to layout patterns.

Despite these advancements, learning-based methods still lack the capability for fine-grained modifications. For instance, when a training set comprises various pattern categories, these methods cannot determine the class of the generated pattern. Editing a specific section of a layout according to particular rules is a frequent necessity, especially with large layouts, to accommodate specific applications [11, 12], but none of existing methods support pattern edition. And the size of generated topology matrix is usually fixed due to the limitation of network architecture and device memory. The fixed pattern size restricts the application of the generated patterns in downstream tasks which requires different pattern size.

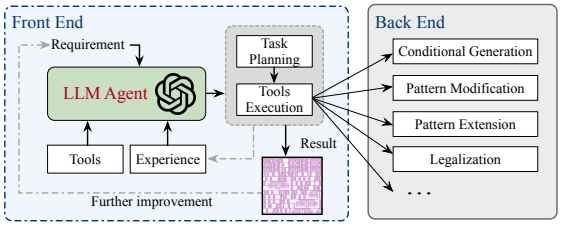


Figure 1: An illustration of ChatPattern.

Additionally, the task of manually synthesizing millions of desired layout patterns using flexible generation tools can be labor-intensive. Pattern library builders must not only master complex generation tools but also comprehend the specialized requirements from downstream users, often communicated in a blend of natural language and professional jargon. Moreover, builders may not always be available or able to respond promptly. To bridge this gap, Large Language Models (LLMs) have proven adept at handling complex tasks across various domains [13, 14]. However, their application as layout pattern library builders remains unexplored.

In response, this paper introduces ChatPattern, an indefatigable layout pattern builder designed to tailor a pattern library to specific requirements articulated in natural language. As illustrated in Figure 1, ChatPattern is composed of two principal components: an expert LLM agent holding design tools and documentation, capable of understanding and executing tasks based on natural language instructions, and a flexible, controllable layout pattern generative model that surpasses existing methods by offering conditional layout generation, precise pattern modification, and unrestricted pattern extension. The contributions of this work are fourfold:

- The presentation of ChatPattern, the inaugural LLM-powered layout pattern generation framework.
- The integration of an expert LLM agent as a pattern library builder, proficient in processing natural language inputs and autonomously operating the necessary tools to fulfill requirements.
- The development of a versatile layout pattern generative model that outperforms existing methods in conditional pattern generation, layout modification, and free-size pattern extension.
- The expansion of the scope of the layout pattern generation task, prompting researchers to focus on more realistic yet challenging tasks such as free-size layout pattern generation.

2 Preliminaries

2.1 The Scope of ChatPattern

ChatPattern is an AI agent that offers a conversational interface, enabling users to use natural language to guide the creation of pattern libraries that meet their specific layout generation needs.

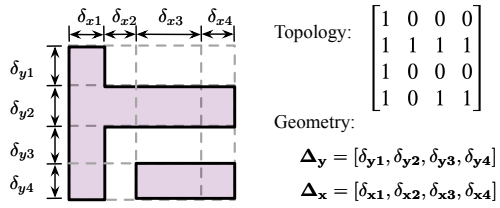


Figure 2: Squish Pattern Representation.

Problem 1 (Target of ChatPattern)**.** Create a framework powered by a large language model that can understand and process natural language instructions, and produce a library of legal layout patterns that meet user needs by utilizing a layout pattern generation tool.

2.2 Fixed-size Layout Pattern Generation

Contemporary research in layout pattern synthesis predominantly focuses on creating patterns of a predetermined dimension. This research domain has witnessed significant contributions that leverage squish-pattern representation [6, 7, 9]. These methods learn to fit the distribution of the binary topology matrix of Squish Pattern [10]. And the generated topology matrix will be further legalized in post-processing to yield a legal layout pattern.

Squish Pattern Representation. A layout pattern, comprising a series of non-overlapping polygons, can be effectively modeled using a squish pattern. The squish pattern is a compact representation of layout pattern that encodes layout topology and geometry into a matrix $\boldsymbol{T}$ and vectors Δ_x, Δ_y. It divides the layout into grids using scan lines along polygon edges, storing intervals in the Δ vectors. Matrix entries are binary, denoting shapes or emptiness. Topology matrix will be further normalized to a fixed-size square for uniformity as introduced in [15]. Figure 2 illustrates how squish pattern works.

Topology Generation via Unconditional Discrete Diffusion. Discrete diffusion model [16] is a type of generative model where the value of every image pixel is limited in a pre-defined discrete space. Similar to normal diffusion models [17, 18], noise is added to input images in forward process, and a model with learnable parameter θ learns to remove noise in reverse process. Given that $x_k \in \{0, 1\}$ is an entry in the topology matrix $\boldsymbol{T}$, 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 of K forward step,

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

where $\mathrm{Cat}(\boldsymbol{x}|\boldsymbol{p})$ denote the categorical distribution of $\boldsymbol{x}$ given $\boldsymbol{p}$. And the forward process can be defined as,

$$\boldsymbol{q}\left(\boldsymbol{x}_k|\boldsymbol{x}_0\right) = \mathrm{Cat}\left(\boldsymbol{x}_k; \boldsymbol{p} = \boldsymbol{x}_0\overline{\boldsymbol{Q}}_k\right), \tag{2}$$

$$\boldsymbol{Q}_k = \begin{bmatrix}1-\beta_k & \beta_k \\ \beta_k & 1-\beta_k\end{bmatrix}, \tag{3}$$

$$\beta_k = \frac{(k-1)\left(\beta_K - \beta_1\right)}{K-1} + \beta_1,\ k = 1, ..., K, \tag{4}$$

where $\overline{\boldsymbol{Q}}_k = \boldsymbol{Q}_1\boldsymbol{Q}_2 \ldots \boldsymbol{Q}_k$ and β_1 and β_K are hyper-parameters.

After sampling a noised topology $\boldsymbol{T}_k$ from the input topology $\boldsymbol{T}_0$ via Equation (2), the model learns to denoise the topology and predicts the logits of the posterior distribution $\boldsymbol{p}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_0|\boldsymbol{x}_k\right)$. Therefore, the k-th reverse step can be calculated as following:

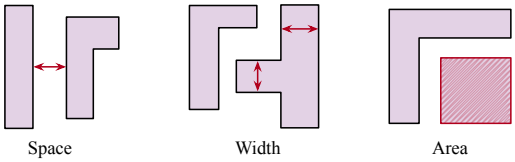


Figure 3: Design rule illustrations. 'Space' means distance between adjacent polygons. 'Width' measures shape size in one direction. 'Area' denotes area of a polygon.

$$\boldsymbol{p}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k\right) = \sum_{\widetilde{\boldsymbol{x}}_0} \boldsymbol{q}\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \widetilde{\boldsymbol{x}}_0\right) \boldsymbol{p}_{\boldsymbol{\theta}}\left(\widetilde{\boldsymbol{x}}_0|\boldsymbol{x}_k\right), \tag{5}$$

where the term $\widetilde{\boldsymbol{x}}_0$ will visit every possible state of $\boldsymbol{x}_0$. Finally, we can denoise a sampled noise $\boldsymbol{T}_K$ and synthesize a topology $\boldsymbol{T}_0$ with the well-trained model by recursively calling the reverse step,

$$p_\theta(\boldsymbol{T}_0|\boldsymbol{T}_K) = p_\theta(\boldsymbol{T}_0|\boldsymbol{T}_1) \prod_{k=2}^{K} p_\theta(\boldsymbol{T}_{k-1}|\boldsymbol{T}_k). \tag{6}$$

Topology Legalization. The generated topology matrix will be further enriched to patterns by matching them with suitable geometry vectors. The generated patterns should satisfy several pre-defined design rules of IC layout [7, 8] as illustrated in Figure 3.

Definition 1 (Legality)**.** We treat a layout pattern as a legal one if the layout pattern is DRC-clean, given the design rules.

$$\text{Legality} = \frac{\#\text{Legal Patterns}}{\#\text{Generated Patterns}} \tag{7}$$

2.3 Free-size Layout Pattern Generation

Compared with fixed-size layout pattern generation task, we aim to synthesize layout patterns with any given size, which is a more challenging task given the limited model output size, device memory, and varied requirements. We adopt the widely-used evaluation metric [6–9], Diversity, to evaluate the quality of generated pattern library. A greater pattern diversity H indicates that the library contains more widely distributed patterns.

Definition 2 (Diversity)**.** The diversity of the patterns library, denoted by H, is defined as the Shannon Entropy of the distribution of the pattern complexities as follows:

$$H = -\sum_i \sum_j P(c_{xi}, c_{yj}) \log P(c_{xi}, c_{yj}), \tag{8}$$

where $P(c_{xi}, c_{yj})$ is the probability of a pattern with complexity (c_{xi}, c_{yj}) sampled from the library, and c_x and c_y are the numbers of scan lines subtracted by one along the x-axis and y-axis, respectively.

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

Problem 2 (Free-size Layout Pattern Extension)**.** Given a set of design rules and target pattern size, the objective of pattern generation is to synthesize a legal pattern library such that the pattern diversity of the layout patterns in the library is maximized.

3 Algorithm

ChatPattern seamlessly integrates a front-end powered by a Large Language Model with a back-end that employs a conditional discrete-diffusion model for layout pattern generation.

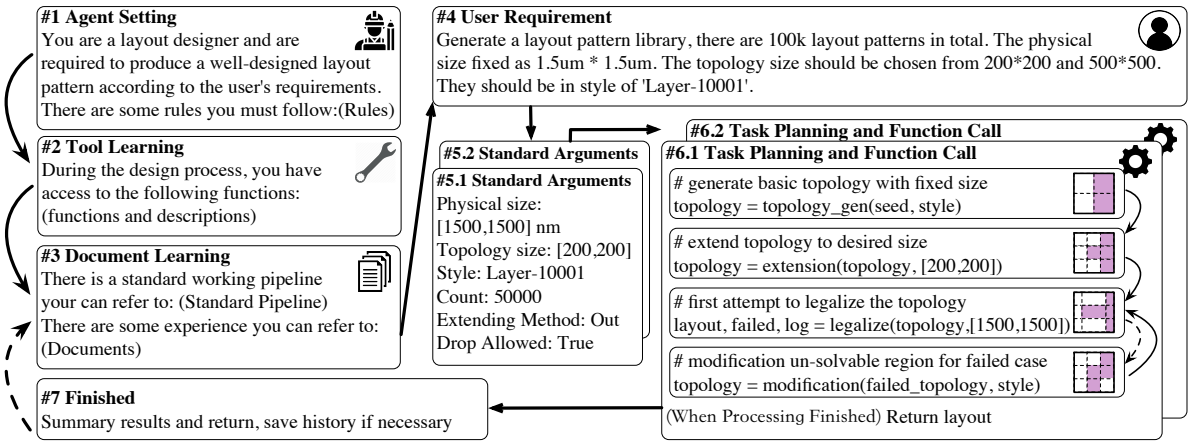


Figure 4: A working pipeline of LLM agent in ChatPattern. Dash lines denote the optional paths.

3.1 Pattern Customization via Expert LLM agent

ChatPattern, leveraging LLM technology, automates pattern library customization. Its primary function is to interpret user requirements expressed in natural language, breaking down complex demands into manageable sub-tasks. These sub-tasks are subsequently processed using specialized pattern generation tools. The detailed operational pipeline of ChatPattern is illustrated in Figure 4.

Requirement Auto-Formatting. Given the complexity of user requests in the Layout Pattern Generation workflow, the LLM agent first translates these requests into a structured format using a predefined template. This template contains all relevant parameters and requirements for utilization by the pattern generation API functions. The template can be summarized by the LLM agent according to the description of functions or provided by the function provider. The arguments can be further divided into required ones, which decide the basic parameter (*e.g.,* pattern count and size) of this task, and optional ones, which are for fine-grained control. An important thing is that one task given by a user can be decomposed into several requirement lists, each for one simple sub-task, to limit the complexity of one task and reduce mistakes. For example, the user request shown in Figure 4 is factorized into two sub-tasks.

Task Planning and Execution. During the last step, ChatPattern identifies and plans the necessary sub-tasks for fulfilling the user's request. For each planned sub-task, ChatPattern schedules a series of structured tasks, which are then addressed using various pattern generation tools like topology generation and legalization. The task planning process is exemplified in Figure 4. For execution, ChatPattern utilizes a Python interpreter or API calls, supported by the layout pattern generation back-end model. Compared with a rule-based arguments-program translator, the LLM agent can address failed cases based on feedback or logs derived from function calls. For example, in the legalization phase, the legalization can easily fail especially for large topology matrices due to the violation of design rules. Applying topology selection and dropping the failed cases can guarantee the legality of the final result, while a lot of effort on topology matrix generation is wasted. Alternative choices for LLM agent are modifying the failed regions with different conditions and trying different initial states, which save time and computation.

Tool Function Learning and Application. The core concept of pattern generation through an LLM agent lies in its ability to operate without directly accessing the generated matrix of zeros and ones. This matrix could surpass the token length limit of the LLM, rendering it incapable of extracting information from such low-level data. The LLM agent instead relies on the route to the end results and comprehension of select overarching characteristics, such as complexity, physical dimensions, or error locations. To construct a pattern library, certain fundamental tools or APIs are indispensable: (1) *Random Topology Generation*, which creates random topologies subject to specific conditions, and (2) *Topology Legalization*, which transforms a topology into a compliant layout pattern. These functions constitute the cornerstone for squish-pattern-based layout pattern generation. Yet, the scope of the generated topology might be constrained by the model size or device capacity, and it might not always conform to the legalization process. Thus, supplementary functions for advanced customization are available: (1) *Topology Extension* augments a topology to a designated size within certain parameters, and (2) *Topology Modification* revises parts of a topology, offering a time-efficient alternative to discarding non-compliant topologies, particularly for expansive patterns. Additional high-level functions may be integrated to bolster the model's capabilities, allowing the LLM agent to capitalize on backend tool enhancements with no need for internal changes.

Learning from Documents and Experience. Documents are reservoirs of high-level knowledge, instrumental in the pattern design process. In the free-size layout pattern generation task, the selection of the extension algorithm plays a pivotal role in determining the quality of the output. Informed algorithm selection can be enhanced through statistical analysis of historical data. Additionally, equipping a model with a standardized operational pipeline at inception can lead to a marked increase in performance. The practice of documenting work history and scrutinizing exceptional cases is equally valuable, laying the groundwork for the model's ongoing refinement.

3.2 Flexible Layout Pattern Generative Model

The pattern generator provides tool support for LLM agent frontend. Our generative model utilizes a conditional discrete diffusion model to fit the topology distributions from multi-source and synthesize topology for further legalization. However, compared with

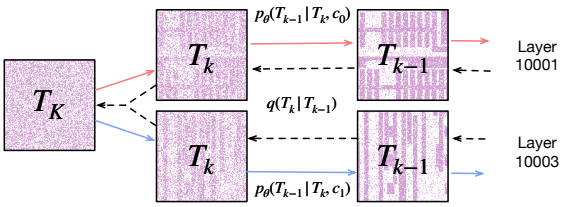


Figure 5: Illustration of conditional pattern generation.

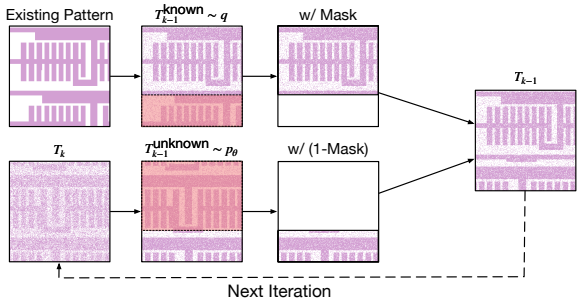


Figure 6: Illustration of pattern modification.

previous work, ChatPattern leverages three novel characters to enable the ability to generate complex topology matrix and meet highly customized requirements.

Property-Conditional Topology Generation. Patterns in a dataset do not always share the design rules or physical properties. For example, the distribution of patterns from the 7-nm manufacturing process is different from that of 130-nm. The diversity of materials and design rules etc. enlarges the difference. Establishing a dataset and training individual models for each kind of patterns is not a sound choice, while directly training a model on the mixture dataset will raise some concerns about the design rules conflict. To address this issue, we add conditions to the reverse process of the diffusion model and the distribution of the generated topology matrix can be specified. Different from the cases in normal image generation, the condition design in pattern generation should consider the design rules, materials, and manufacturing process. The k-th step of distribution specified reverse process can be defined as,

$$\boldsymbol{p_\theta}\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, c\right) = \sum_{\widetilde{\boldsymbol{x}}_0} \boldsymbol{q}\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \widetilde{\boldsymbol{x}}_0\right) \boldsymbol{p_\theta}\left(\widetilde{\boldsymbol{x}}_0|\boldsymbol{x}_k, \boldsymbol{c}\right), \tag{9}$$

where $\boldsymbol{c}$ denotes the conditions. An illustration of the training and sampling process is shown in Figure 5. During the training process, the model optimizes the loss function,

$$L = D_{\text{KL}}\left(\boldsymbol{q}\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \boldsymbol{x}_0\right) \parallel \boldsymbol{p}_\theta\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \boldsymbol{c}\right)\right) - \lambda \log \boldsymbol{p_\theta}\left(\boldsymbol{x}_0|\boldsymbol{x}_k, \boldsymbol{c}\right). \tag{10}$$

When the training process is finished, a topology matrix with condition $\boldsymbol{c}$ can be generated by a K-step reverse process,

$$p_\theta(\boldsymbol{T}_0|\boldsymbol{T}_K, \boldsymbol{c}) = p_\theta(\boldsymbol{T}_0|\boldsymbol{T}_1, \boldsymbol{c}) \prod_{k=2}^{K} p_\theta(\boldsymbol{T}_{k-1}|\boldsymbol{T}_k, \boldsymbol{c}), \tag{11}$$

where $\boldsymbol{T}_K$ is a randomly-sampled noise.

Pattern Modification. Given an existing pattern topology matrix $\boldsymbol{T}_0^{\text{known}}$, making modifications to any desired region on it can be useful when dealing with failed topology. We denote kept pixels as $\boldsymbol{M} \odot \boldsymbol{T}_0^{\text{known}}$ and denote the masked pixels as $(1-\boldsymbol{M}) \odot \boldsymbol{T}_0^{\text{known}}$. Since in every reverse step of Equation (11), $\boldsymbol{T}_{k-1}$ depends solely on

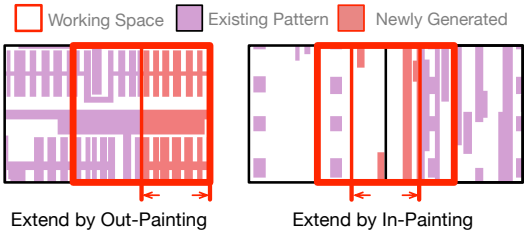


Figure 7: Pattern extension via In-Painting and Out-Painting.

$\boldsymbol{T}_k$, we replace the kept region in $\boldsymbol{T}_k$ by the noised pixels in given topology, $\boldsymbol{M} \odot \boldsymbol{T}_k^{\text{known}}$. The noised topology $\boldsymbol{T}_k^{\text{known}}$ is obtained by the forward process in Equation (2). And $(1-\boldsymbol{M}) \odot \boldsymbol{T}_{k-1}^{\text{unknown}}$ will be generated by the model with the condition of $\boldsymbol{M} \odot \boldsymbol{T}_k^{\text{known}}$. And different with general image generation[17, 19], the modification of patterns should consider the design rules and the condition $\boldsymbol{c}$ should match the distribution of given topology matrix $\boldsymbol{T}_0^{\text{known}}$,

$$\begin{aligned} \boldsymbol{T}_{k-1}^{\text{known}} &\sim \boldsymbol{q}\left(\boldsymbol{T}_{k-1}|\boldsymbol{T}_0^{\text{known}}\right), \\ \boldsymbol{T}_{k-1}^{\text{unknown}} &\sim \boldsymbol{p_\theta}\left(\boldsymbol{T}_{k-1}|\boldsymbol{T}_k, \boldsymbol{c}\right), \\ \boldsymbol{T}_{k-1} &= \boldsymbol{M} \odot \boldsymbol{T}_{k-1}^{\text{known}} + (1-\boldsymbol{M}) \odot \boldsymbol{T}_{k-1}^{\text{unknown}}, \end{aligned} \tag{12}$$

where $\boldsymbol{T}_0^{\text{known}}$ shares the design rules with patterns in condition $\boldsymbol{c}$. An illustration of pattern modification can be found in Figure 6.

Pattern Extension. Extending a given pattern to a larger one is a practical function since the model output usually takes a fixed size while the required patterns can vary among a large range. Pattern extension can be achieved via In-Painting and Out-Painting with pattern modification techniques. An illustration of In-Painting extension and Out-Painting extension can be found in Figure 7. By modifying the adjacency border and corner of a concatenated topology matrix, we can merge the shape from both sides and synthesize a larger topology matrix, which we denoted as In-Painting. We treat the extending method that directly generates a new border of an existing pattern as Out-Painting.

By recursively extending a given pattern, we can extend the pattern to any desired size without considering memory limitation, since only the region in working space will be taken into computation. If we assume the target size of topology is $[W, H]$ and the window size of the model is $[L, L]$, the number of sampling for In-Painting can be calculated by, $N_{\text{in}} = (2\lceil \frac{W}{L} \rceil - 1)(2\lceil \frac{H}{L} \rceil - 1)$. With a stride S, the number of sampling for Out-Painting is $N_{out} = (\lceil \frac{W-L}{S} \rceil + 1)(\lceil \frac{H-L}{S} \rceil + 1)$.

Legalization. We utilize the non-linear legalization function proposed in DiffPattern[9] and denoted it as,

$$\text{Legalization}(\cdot) = f_{\mathcal{R}}(\boldsymbol{F}, \boldsymbol{T}), \tag{13}$$

where $\mathcal{R}$ is the set of design rules, $\boldsymbol{T}$ is the topology and $\boldsymbol{F}$ is the physical size of layout pattern. When the legalization fails, the unreasonable region can be located and returned due to the explainable feature of the legalization function.

4 Experimental Results

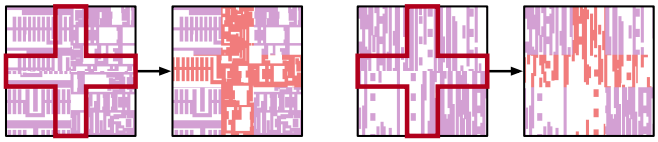
4.1 Layout Pattern Generation

Datasets. We follow previous works [8, 9] to obtain the dataset of small layout pattern images with the size of 2048×2048 nm^2 by splitting the layout map from ICCAD contest 2014. The size of the

Table 1: Comparison on Legality and Diversity on legal patterns. '/' refers to not applicable.

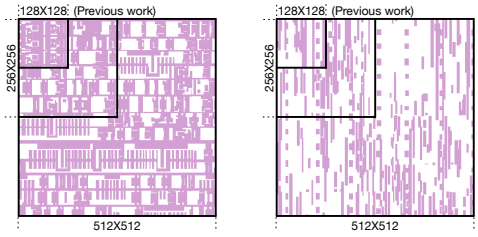
Task	Set/Method	Training Set*	Size	Layer-10001		Layer-10003		Total†	
				Legality (↑)	Diversity (↑)	Legality (↑)	Diversity (↑)	Legality (↑)	Diversity (↑)
Fixed-size	Real Patterns	/	128^2	/	10.731	/	8.769	/	10.625
	CAE+LegalGAN [7]	Layer-10001		3.74%	5.814	/	/	/	/
	VCAE+LegalGAN [7]	Layer-10001		84.51%	9.867	/	/	/	/
	LayouTransformer [8]	Layer-10001		89.73%	10.527	/	/	/	/
	DiffPattern [9]	Layer-10001/10003		**99.97%**	10.711	99.98%	8.578	**99.98%**	10.633
	ChatPattern	Layer-10001/10003		**99.97%**	**10.796**	**99.99%**	**8.625**	**99.98%**	**10.650**
Free-size	Real Patterns	/	256^2	/	12.702	/	10.696	/	12.695
	[9] w/ Concatenation	Layer-10001/10003		57.78%	10.719	93.69%	10.511	75.74%	11.706
	ChatPattern	Layer-10001/10003		**87.36%**	**11.154**	**99.78%**	**10.556**	**93.57%**	**11.830**
	Real Patterns	/	512^2	/	13.435	/	12.139	/	13.787
	[9] w/ Concatenation	Layer-10001/10003		0.29%	5.714	40.83%	11.555	20.56%	11.359
	ChatPattern	Layer-10001/10003		**36.42%**	**10.401**	**98.86%**	**11.620**	**67.64%**	**12.133**
	Real Patterns	/	1024^2	/	13.573	/	12.644	/	14.109
	[9] w/ Concatenation	Layer-10001/10003		0.00%	0.000	0.64%	6.926	0.32%	6.926
	ChatPattern	Layer-10001/10003		**1.19%**	**6.438**	**94.96%**	**11.981**	**47.80%**	**11.992**

* All training datasets are the 128×128 version.
† We collected generated samples from both Layer-10001/10003 and evaluated them together.



(a) Layer-10001 style (b) Layer-10003 style

Figure 8: Example of 256×256 topology matrix generated by In-Painting (without legalization).



(a) Layer-10001 style (b) Layer-10003 style

Figure 9: 512×512 topology matrix generated by Out-Painting.

extracted topology matrix is fixed as 128×128 in squish pattern representation in the baseline setting. There are two different style layouts, denoted as Layer-10001 and Layer-10003. Layer-10001 is widely used in previous methods and we introduce layer-10003 to evaluate the model ability of style specification. Furthermore, by splitting the layout map into 4×, 16× and 64× larger size with overlap, we also obtained 4×, 16× and 64× larger topology matrix in squish pattern representation correspondingly.

Diffusion Model Configuration. Following the common settings [16, 17], we use a U-Net [20] as our backbone in the conditional discrete diffusion model. The implementation of U-net follows that in [17]. To extract the embedding of the condition, we use a stack of 3 linear layers. The condition embedding is added into the embedding of the time step to control the style of the generated patterns.

Training & Testing Details. To keep the same size with previous squish-pattern-based methods [7, 9], we train ChatPattern on the 128×128 union-datasets (Layer-10001 and Layer-10003) with class conditions for 1.0M iterations with a batch size of 128. The network is optimized by an Adam optimizer with a learning rate 2e-4. Drop out is 0.1 the grad clip is set to 1, and the loss coefficient λ is set to 1e-3. Diffusion length $K = 1000$ and the noise schedule β_k is linearly increased from 0.01 to 0.5. The training procedure takes about 250 GPU hours in total. The back-end of ChatPattern is trained on 128×128 dataset and the size of directly generated typologies is fixed at 128×128. The topology matrix sampled from the network will be further legalized as explained in Section 3.2. We have noticed previous work [9] applies topology selection and pushes the legality to 100%. Since every squish-pattern-based method can reach 100% legality via selection, we remove the selection step from all methods when evaluating their performance to compare the model directly. We further forbid our pattern modification function calling in both fixed-size and free-size pattern generation tasks to have a fair comparison.

Fixed-size Pattern Generation. We compare our method with previous layout pattern generation methods, CAE [6], VCAE [7], LegalGAN [7] and LayouTransformer [8] on the widely used benchmark 128×128 Layer-10001. And we further re-implemented the previous SOTA, DiffPattern[9], on Layer-10001 and Layer-10003 individually since directly mixing patterns from different distributions can easily lead to a conflict for [9], as we discussed in Section 3.2. We report the result on 10,000 samples generated by ChatPattern for each class. The legality and the diversity of legal patterns on each dataset are reported in Table 1. According to the results, on the 128×128 pattern generation task, ChatPattern gets a reasonable improvement compared with previous SOTA thanks to its ability to utilize a multi-source training dataset. However, both ChatPattern and the existing models have already fitted the distribution of real patterns well in the simple 128×128 benchmarks.

Free-size Pattern Generation. ChatPattern is specialized in free-size pattern generation, a task highly applicable to real-world scenarios. For this task, we established three distinct experimental size settings to quantitatively assess the model. The sizes of the target topology matrices ranged from 256^2 to 1024^2. We generated 10,000 samples for each class across all size levels, examining their Legality

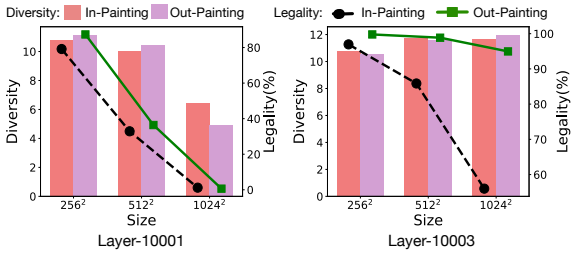


Figure 10: Evaluation of In-Painting and Out-Painting.

and Diversity as we did in the fixed-size pattern generation tasks. Additionally, patterns extracted from real datasets served as references and are duly noted in the results table. The baseline is the SOTA in fixed-size pattern generation, DiffPattern [9]. To create larger patterns, the baseline method can only stitches together small patches of the same class, but the method often breaches design rules. For instance, when patch size reaches 512^2, the legality of patterns generated by DiffPattern with concatenation plummets to nearly zero (0.29%) and under half (40.83%) for the Layer-10001 and Layer-10003 datasets, respectively. Our tests with ChatPattern cover all the aforementioned settings. ChatPattern leverages its extension function, utilizing both out-painting and in-painting algorithms, to synthesize high-quality patterns. We have illustrated some instances in Figure 8 and Figure 9. The LLM agent is initialized with a predefined working pipeline, as depicted in Figure 4. The accompanying documentation includes statistical data on pattern extension through various algorithms, as seen in Figure 10. This documentation provides an insight that out-painting typically yields better legality, while in-painting excels in diversity under certain conditions. The user's request was simply to generate 10,000 patterns for each experimental setting. Results tabulated in Table 1 demonstrate ChatPattern's superiority over the baseline, particularly in the demanding task of generating large patterns of sizes 512^2 and above.

4.2 Evaluation of LLM agent

We take the user requirement in Figure 4 as a running example to show the LLM agent's ability on requirement auto-formatting and unseen mistake-processing.

Requirement Auto-formatting. The LLM agent is required to fill out a standard requirement list to make sure the target of every sub-task is specific and can be handled within a simpler script. An example of the requirement list is following.

> # Requirement - subtask 1
> ## Basic Part: Topology Size: [200, 200], Physical Size: [1500, 1500] nm, Style: Layer-10001, Count: 50000,
> ## Advanced Part: Extension Method: Out (Default: Out), Drop Allowed: True (Default: True), Time Limitation: None (Default: None).

Unseen Mistake-processing. As explored in Section 3, the generation of extensive topology matrices poses significant time demands. Should the legalization process fail, the function earmarks the errors within a log. We present a case where the LLM agent adeptly navigates unanticipated mistakes during legalization. It is noteworthy that such error-handling expertise is not pre-coded within the standard operational procedures or the experiential documentation. This instance serves to highlight the LLM agent's intrinsic capacity to address unforeseen complications.

> % In previous attempts, one pattern can't pass the legalization.
> **Thought**: Since legalization has failed twice in the same region, I will try to in-paint that specific area with the same style and then attempt legalization again.
> **Action**: Topology_Modification
> **Action Input**: "topology_path":${path}, "upper": 12, "left": 56, "bottom": 33, "right": 73, "style": "Layer-10001", "seed": 42
> % Continue Processing

5 Conclusion

In this paper, we introduced ChatPattern, a novel framework for pattern generation utilizing a LLM. ChatPattern provides a user-friendly interface that accepts natural language inputs to tailor the pattern library to specific needs. And ChatPattern enables a flexible pattern generation back-end and extends the task of pattern generation to a more demanding yet challenging setting, free-size pattern generation. Nonetheless, ChatPattern still lacks global guidance when generating large patterns and can not handle complex oversized patterns well, which we left for future exploration.

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*, 2014, pp. 1–6.

[2] J. Kuang and E. F. Young, "An efficient layout decomposition approach for triple patterning lithography," in *DAC*, 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*, 2018, pp. 1–6.

[6] 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.

[7] X. Zhang, J. Shiely, and E. F. Young, "Layout pattern generation and legalization with generative learning models," in *ICCAD*, 2020, pp. 1–9.

[8] 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*, 2022.

[9] Z. Wang, Y. Shen, W. Zhao, Y. Bai, G. Chen, F. Farnia, and B. Yu, "Diffpattern: Layout pattern generation via discrete diffusion," *arXiv preprint arXiv:2303.13060*, 2023.

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

[11] S. Sun, Y. Jiang, F. Yang, B. Yu, and X. Zeng, "Efficient hotspot detection via graph neural network," in *DATE*, 2022, pp. 1233–1238.

[12] W. Zhao, X. Yao, Z. Yu, G. Chen, Y. Ma, B. Yu, and M. D. Wong, "Adaopc: A self-adaptive mask optimization framework for real design patterns," in *ICCAD*, 2022, pp. 1–9.

[13] G. L. Xuanhe Zhou, Zhaoyan Sun, "Db-gpt: Large language model meets database," 2023.

[14] Z. He, H. Wu, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, "Chateda: A large language model powered autonomous agent for eda," *arXiv preprint arXiv:2308.10204*, 2023.

[15] 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.

[16] 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.

[17] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *NeurIPS*, vol. 33, pp. 6840–6851, 2020.

[18] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," in *ICLR*, 2020.

[19] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. Van Gool, "Repaint: Inpainting using denoising diffusion probabilistic models," in *CVPR*, 2022, pp. 11 461–11 471.

[20] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *MICCAI*. Springer, 2015, pp. 234–241.