WHERE **INNOVATION** BEGINS

# DESIGN AUTOMATION CONFERENCE

**JULY 9–13, 2023**

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

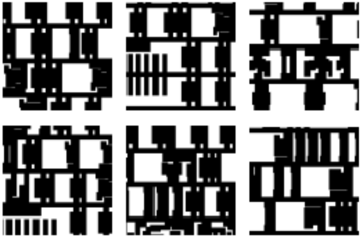
# DiffPattern: Layout Pattern Generation via Discrete Diffusion

Zixiao Wang[1], Yunheng Shen[2], Wenqian Zhao[1], Yang Bai[1], Guojin Chen[1], Farzan Farnia[1], **Bei Yu**[1]

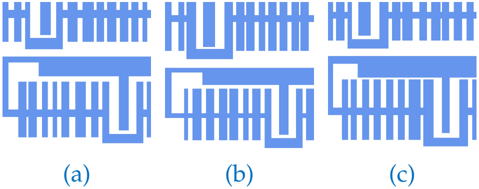
[1]Chinese University of Hong Kong
[2]Tsinghua University

# Background Knowledge

# Layout Pattern Generation



Original Layout Patterns [ICCAD’20]



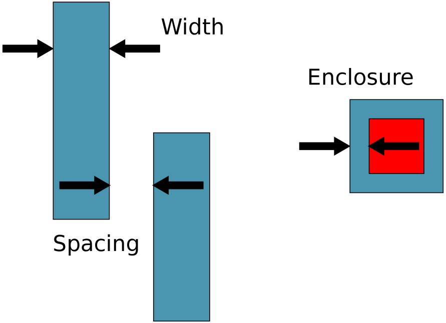
Generated Layout Patterns (Ours)

VLSI layout patterns provide critical resources in various designs for manufacturability research, from early technology node development to back-end design and sign-off flows[DAC’19][1].

[1] H. Yang *et al.*, “Deepattern: Layout pattern generation with transforming convolutional auto-encoder”, in *DAC*, 2019, pp. 1–6.

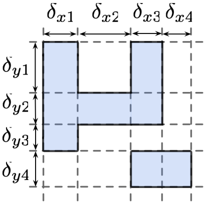
# An End-to-End Learning Solution?

**The three basic DRC checks**



- Maybe No
- Gap between Discrete Rules and Continuous DNN Model

# Squish Pattern Representation



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

Geometry: $\Delta_x = [\delta_{x1}, \delta_{x2}, \delta_{x3}, \delta_{x4}]$

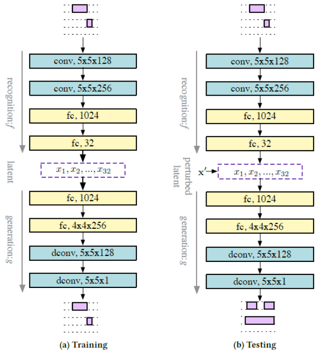
$\Delta_y = [\delta_{y1}, \delta_{y2}, \delta_{y3}, \delta_{y4}]$

## Squish Pattern [US Patent'14][2]

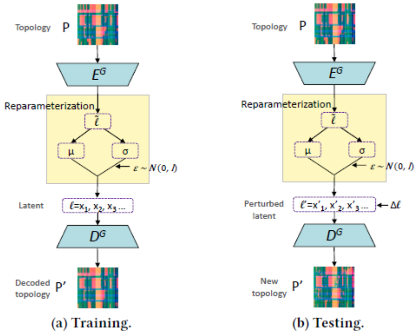
- Lossless and efficient representation method
- Encodes layout into pattern topology matrix and geometric information
- Problem #1: information density of each pixel is still not satisfactory

[2] F. E. Gennari and Y.-C. Lai, *Topology design using squish patterns*, US Patent 8,832,621, Sep. 2014.

# Novel Pattern Generation



(a) Encoder-Decoder [DAC'19]

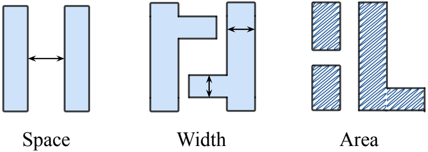


(b) VAE-based [ICCAD'20][3]

- Generate gray image (topology) and transfer it into a binary image
- May lead to a deduction of information
- Problem #2: How to generate a binary mask directly?

[3] X. Zhang *et al.*, "Layout pattern generation and legalization with generative learning models", in *Proc. ICCAD*, 2020, pp. 1–9.

# Pattern Legalization



Examples of DRC Rule

## Finding legal distance vector for each topology

- Solving a Linear System (1D pattern) [DAC'19].
- Using Exist Distance Vector (2D pattern) [ICCAD'20]
- Problem #3: 2D pattern introduces non-linear constraint, hard to solve!

# Evaluation

- Pattern Diversity. Shannon entropy of the pattern complexity.

$$H = -\sum_{i}\sum_{j} P(c_{xi}, c_{yj}) \log P(c_{xi}, c_{yj}), \tag{1}$$

- Pattern Legality.

$$L = \frac{\#\ \text{Legal Patterns}}{\#\ \text{All Patterns}}. \tag{2}$$

# Denoising Diffusion Probabilistic Models [NeurIPS'20][4]

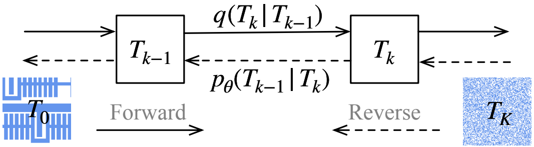


Illustration of denoising diffusion process.

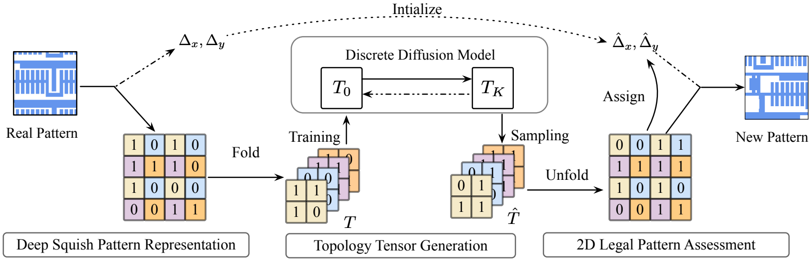
Forward Process: $\boldsymbol{q}\left(\boldsymbol{T}_k|\boldsymbol{T}_{k-1}\right) := \mathcal{N}\left(\boldsymbol{T}_k; \sqrt{1-\beta_k}\boldsymbol{T}_{k-1}, \beta_k\boldsymbol{I}\right).$

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

[4] J. Ho *et al.*, "Denoising diffusion probabilistic models", *NeurIPS*, vol. 33, pp. 6840–6851, 2020.

# Proposed Method: DiffPattern

# Overview



Intialize

$\Delta_x, \Delta_y$

$\hat{\Delta}_x, \hat{\Delta}_y$

Real Pattern

Discrete Diffusion Model

$T_0$

$T_K$

Training

Sampling

Fold

Unfold

Assign

New Pattern

$T$

$\hat{T}$

| | | | |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 |

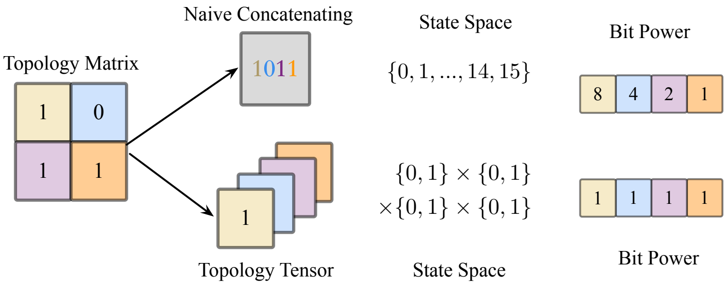
Deep Squish Pattern Representation

Topology Tensor Generation

2D Legal Pattern Assessment

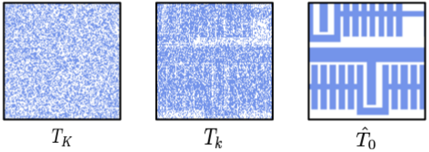
An illustration of the Diffpattern framework for reliable layout pattern generation.

# Problem #1: Deep Squish Pattern Representation



- The Topology Tensor is a lossless and compact representation of the topology matrix.
- The Naive Concatenating brings unbalanced power to each bit and an exponentially increasing state space.

# Problem #2: Topology Tensor Generation



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

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

Multiple step forward at once. $\boldsymbol{q}(\boldsymbol{x}_k|\boldsymbol{x}_0) = Cat(\boldsymbol{x}_k; \boldsymbol{p} = \boldsymbol{x}_0\bar{\boldsymbol{Q}}_k), \quad \bar{\boldsymbol{Q}}_k = \boldsymbol{Q}_1\boldsymbol{Q}_2...\boldsymbol{Q}_k$

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

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

# Problem #2: Topology Tensor Generation

A uniform stationary distribution is a natural choice in topology tensor generation. Given any $\boldsymbol{x}_0$, the distribution of every entry $\boldsymbol{x}_k$ should follows,

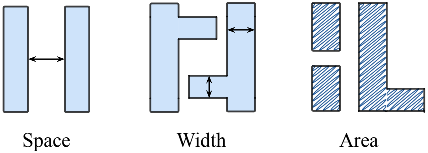
$$\boldsymbol{q}(\boldsymbol{x}_k|\boldsymbol{x}_0) \rightarrow [0.5, 0.5]\,, \text{ when } k \rightarrow K. \tag{3}$$

$$\boldsymbol{Q}_k = \begin{bmatrix} 1-\beta_k & \beta_k \\ \beta_k & 1-\beta_k \end{bmatrix}, \tag{4}$$

$$\beta_k = \frac{(k-1)\left(\beta_K - \beta_1\right)}{K-1} + \beta_1,\ k = 1, ..., K, \tag{5}$$

where $\beta_1$ and $\beta_K$ are hyperparameters.

# Problem #3: 2D Pattern Legalization



Examples of DRC Rule

$$\begin{cases} \delta_{xi}, \delta_{yj} > 0, & \forall \delta_{xi}, \delta_{yj}; \\ \sum \delta_{xi} = \sqrt{C}M, \quad \sum \delta_{yj} = \sqrt{C}M; & \\ \sum_{i=a}^{b} \delta_i \geq Space_{min}, & \forall (a,b) \in Set_S; \\ \sum_{i=a}^{b} \delta_i \geq Width_{min}, & \forall (a,b) \in Set_W; \\ \sum \delta_{xi}\delta_{yj} \in [Area_{min}, Area_{max}], & \forall \text{ Polygon}; \end{cases} \tag{6}$$

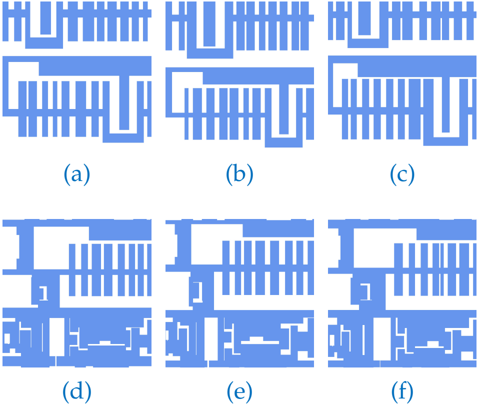
# Experiment Results

# Diversity and Legality

| Set/Method | Generated Topology | Generated Patterns | | Legal Patterns | |
|---|---|---|---|---|---|
| | | Patterns | Diversity (↑) | Legality (↑) | Diversity (↑) |
| Real Patterns | - | - | - | 13869 | 10.777 |
| CAE [DAC'19] | 100000 | 100000 | 4.5875 | 19 | 3.7871 |
| VCAE [ICCAD'20] | 100000 | 100000 | **10.9311** | 2126 | 9.9775 |
| CAE+LegalGAN [ICCAD'20] | 100000 | 100000 | 5.8465 | 3740 | 5.8142 |
| VCAE+LegalGAN [ICCAD'20] | 100000 | 100000 | 9.8692 | 84510 | 9.8669 |
| LayouTransformer [ICCAD'22] | - | 100000 | 10.532 | 89726 | 10.527 |
| DiffPattern-S | 100000 | 100000 | 10.815 | **100000** | **10.815** |
| DiffPattern-L | 100000 | 10000000 | 10.815 | **10000000** | **10.815** |

- DiffPattern achieves a perfect performance (i.e. 100%) under the metric of legality.
- DiffPattern also gets reasonable improvement (10.527→10.815) on the diversity.
- We generate 100 different layout patterns from each topology in DiffPattern-L.

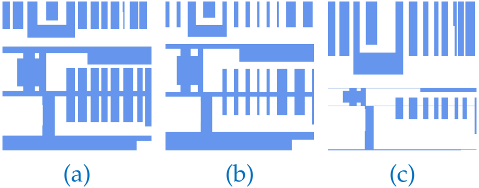
# Flexibility: Generate Different Patterns from Single Topology.


(a) (b) (c)

(d) (e) (f)

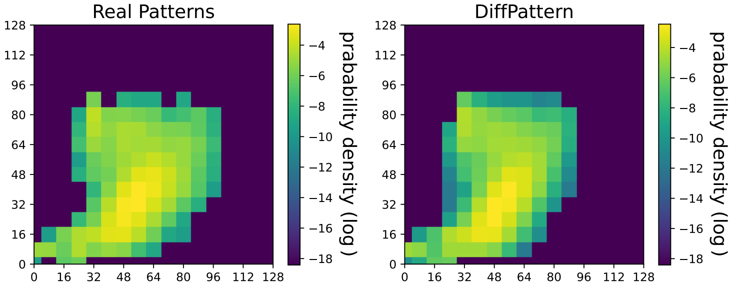
Different layout patterns that are generated from a single topology with the same design rule.

# Flexibility: Generate Legal Patterns with Different Design Rules.



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

# Distribution of Complexity



An illustration of complexity distribution.

# Model Efficiency

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

- Initializing with existing results achieves 2.30× acceleration on CPU.

WHERE
INNOVATION
BEGINS

DESIGN
AUTOMATION
CONFERENCE

# THANK YOU!