CODEPAINT: GENERATING IC LAYOUTS IN A LEARNED DISCRETE SPACE

Anonymous authors
Paper under double-blind review

ABSTRACT

Early design-technology co-optimization requires both rule-clean layout patterns that satisfy design rules and rule-targeted patterns with specified critical dimensions. Meeting these requirements is challenging because small geometric variations can violate design rules, while sampling from a pretrained natural image distribution offers limited controllability. We present CodePaint, a framework for controllable IC layout generation in a learned discrete space. A vector-quantized vocabulary captures local layout features, and absorbing diffusion completes their spatial configurations through one-way token updates that preserve context and revealed tokens. For rule-targeted generation, we introduce TokenProbe, which selects tokens to retain during completion by probing encoder responses to geometric perturbations, without additional training. On ASAP7, CodePaint achieves 75.69% direct DRC compliance compared with 12.00% for PatternPaint, rising to 78.38% after cleanup. TokenProbe raises the valid target realization rate to 22.5%, up from 14.7% for the fixed-block baseline. On a commercial 16 nm PDK, CodePaint adapted with only 20 target clips outperforms PatternPaint adapted with 100 clips. Together, these results show that a discrete prior learned directly from layout geometry supports rule-clean generation, rule-targeted generation, and sample-efficient PDK adaptation.

1 INTRODUCTION

Synthetic layout pattern libraries are a cornerstone of early design-technology co-optimization (DTCO) and process design kit (PDK) development. Before real designs exist, engineers must qualify hundreds of design rules (DRs) that continue to change as a technology node matures, and every new rule set needs a diverse pattern library to support downstream tasks such as optical proximity correction (OPC) (Gao et al., 2014; Jiang et al., 2019; 2022; Hamouda et al., 2017), hotspot detection (Reddy et al., 2019; Chen et al., 2019), and design rule manual qualification (Kabeel et al., 2023). These tasks do not merely need patterns that are rule-clean; they need patterns that *exercise* a chosen rule at a target critical dimension, for example a tip-to-side spacing placed exactly at a candidate rule value, so that the rule's process window can be measured before it is frozen. Writing rule-based generators for each rule set demands substantial engineering effort and must be redone at every rule change, and advanced technology nodes bring increasingly numerous and complex design rules (Synopsys), which is why learning-based pattern generation has attracted growing interest.

PatternPaint (Zhou et al., 2025b) demonstrated that diffusion-based inpainting can generate legal layout patterns for an advanced sub-2 nm industrial PDK under full sign-off design-rule checking using only 20 starter patterns. It rasterizes layouts as images and fine-tunes a continuous latent diffusion model pretrained on natural images. However, its decoded outputs exhibit pixel noise and boundary artifacts, and its reported evaluation yields no DRC-clean outputs without template-based denoising. Although this correction suppresses artifacts, snapping generated boundaries to the starter template can also change critical dimensions, complicating precise geometric control. On the same commercial PDK, Vario reports improved DRC compliance with more training data, yet achieves only a 58.7% pass rate even with 10,000 training patterns (Zhou et al., 2025a;c), leaving substantial room for improving geometric validity.

These limitations raise two coupled questions: *how should rule-critical geometry be represented* and *how should a generative model leverage this representation to enable valid and controllable layout*

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

generation? Unlike generic image generation, chip layout generation poses a distinct challenge: small boundary displacements can change widths and spacings enough to violate design rules, even when the patterns remain visually similar (Appendix F.4). Validity alone is insufficient for DTCO, where technology development involves *probing*: systematically varying a candidate rule and regenerating patterns at each value to evaluate its effect on area, yield, or printability (Kagalwalla et al., 2012; Grubbs et al., 2025). Generated patterns must therefore realize specified critical dimensions, including targets rarely represented in the training data.

To address this question, we propose CodePaint, a codebook-based absorbing discrete diffusion inpainting layout generator. A vector-quantized tokenizer maps layouts to compact spatial token grids (van den Oord et al., 2017), and absorbing diffusion learns to complete masked regions while preserving observed tokens (Austin et al., 2021). This formulation models configurations of learned layout features and provides a natural interface for geometric conditioning through retained tokens. For rule-targeted generation, we instantiate the requested dimension in a seed layout and introduce *TokenProbe* to select which tokens to retain during neighborhood completion. TokenProbe probes encoder responses to controlled geometric perturbations to identify tokens sensitive to the target dimension, enabling inference-time geometric conditioning without additional training. Finally, we adapt the pretrained layout prior to new PDKs using limited target data, supporting early-stage technology development.

We evaluate CodePaint using Calibre DRC (Mentor Graphics, 2018) after stitching generated geometry back into GDSII layouts. On ASAP7 (Clark et al., 2016), CodePaint achieves 75.69% direct DRC compliance, compared with 12.00% for PatternPaint, with post-cleanup rates of 78.38% and 42.62%, respectively. To demonstrate geometric control, we evaluate multiple spacing targets where validity requires no additional DRC violations relative to the encode–decode reference. TokenProbe improves valid target realization over BoxKeep at a matched frozen-token budget, from 14.7% to 22.5%. On a commercial 16 nm PDK, CodePaint adapted with only 20 target clips outperforms PatternPaint in both direct and post-cleanup compliance. Together, these results demonstrate improved geometric validity, effective target control, and sample-efficient adaptation. Concretely, CodePaint contributes:

- **A learned discrete layout representation.** We demonstrate that vector quantization provides an effective representation for chip layouts, supporting higher direct DRC compliance than the evaluated continuous representations and alternative discrete tokenizers.
- **Absorbing diffusion for layout completion.** We formulate layout generation as progressive completion with one-way token updates, preserving context and preventing subsequent corruption of revealed tokens. This mechanism substantially suppresses pixel artifacts and improves geometric validity relative to other transitions, reducing reliance on post-hoc cleanup.
- **Training-free geometric control.** We propose *TokenProbe*, a token-retention strategy for rule-targeted generation. TokenProbe selects tokens through encoder responses to geometric perturbations, improving valid target realization over fixed-block retention at a matched token budget.
- **Sample-efficient cross-PDK adaptation.** We adapt the pretrained layout prior through decoder refinement, geometric augmentation, and topology-balanced denoiser fine-tuning, demonstrating effective transfer to a commercial PDK with as few as 20 target clips.

2 Preliminaries

2.1 Learned Representations

Learned continuous or discrete representations affect information retention and diffusion quality (Rombach et al., 2022; Zhong et al., 2026; Yuan et al., 2026). Graphic-layout and CAD methods organize representations around elements or primitives (Inoue et al., 2023; Zhang et al., 2023; Xu et al., 2022; 2024). We instead learn from rasterized IC layouts.

Vector quantization. A vector-quantized variational autoencoder (VQ-VAE) (van den Oord et al., 2017) uses an encoder E, a decoder D, and a codebook $\mathcal{E} = \{\mathbf{e}_k\}_{k=1}^{K} \subset \mathbb{R}^d$. Each spatial feature

of $\mathbf{h} = E(x) \in \mathbb{R}^{H' \times W' \times d}$ is assigned to its nearest codebook entry:

$$z_i = \underset{k \in \{1,\dots,K\}}{\arg\min} \|\mathbf{h}_i - \mathbf{e}_k\|_2^2, \qquad \mathbf{z}_{q,i} = \mathbf{e}_{z_i}. \tag{1}$$

The index grid $\mathbf{z} \in \{1, \dots, K\}^{H' \times W'}$ provides categorical tokens, while the embedding grid $\mathbf{z}_q$ is decoded as $\hat{x} = D(\mathbf{z}_q)$. Training combines reconstruction, codebook, and commitment losses, using straight-through gradients through quantization. The learned prototypes provide a data-adaptive discretization of encoder features, which we use for layout tokenization.

Alternative discretizations. Finite scalar quantization (FSQ) (Mentzer et al., 2024) uses straight-through rounding on fixed scalar grids, with L_j levels per coordinate and vocabulary size $K = \prod_j L_j$. Straight-through Gumbel-Softmax (Jang et al., 2017) selects categories from encoder logits using hard forward samples and a temperature-controlled soft relaxation for backpropagation. We compare these discretization mechanisms in Table 1 to assess their effects on geometric fidelity and downstream layout generation under a common absorbing-state diffusion framework.

2.2 Generative Modeling and Control

Given a representation, generation models the distribution of its spatial configurations. Prior layout pattern generators, from early learned models (Yang et al., 2022; Zhang et al., 2020; 2021) to discrete diffusion over squish topologies (Wang et al., 2023; 2024b; 2025), synthesize only the topology and delegate physical dimensions to separate rule-constrained solvers. This decomposition requires translating every design rule into solver-interpretable constraints, which is feasible for simplified academic rule sets but fails to scale to industrial decks with discrete width values and width-dependent spacing (Zhou et al., 2025a;b). To circumvent the solver, PatternPaint (Zhou et al., 2025b) adopts a pixel representation whose coordinates carry fixed physical dimensions, letting the diffusion model learn design rules directly from data and generate geometry jointly with topology. It performs layout inpainting with a pretrained continuous latent diffusion model (Rombach et al., 2022), whose lossy decoder introduces noise along polygon edges, and therefore relies on template-based geometric post-processing to recover clean boundaries. We retain raster inputs and outputs, but perform absorbing-state diffusion over discrete indices in a learned layout codebook rather than continuous latent variables, yielding substantially cleaner boundaries without relying on geometric post-processing.

Absorbing-state diffusion. Discrete diffusion and iterative masked-token prediction enable image generation in learned token spaces (Gu et al., 2022; Chang et al., 2022). Discrete denoising diffusion probabilistic models (D3PMs) (Austin et al., 2021) define a forward corruption process over categorical states and learn the corresponding reverse process. In the absorbing-state variant, the state space consists of K ordinary tokens and a special mask token m. With states ordered as $(1, \dots, K, m)$, the row-stochastic forward transition matrix is

$$\mathbf{Q}_t^{\text{abs}} = \begin{pmatrix} (1-\beta_t)\mathbf{I}_K & \beta_t \mathbf{1}_K \\ \mathbf{0}_K^\top & 1 \end{pmatrix}, \tag{2}$$

where $\beta_t \in [0, 1]$ is the per-step masking probability, $\mathbf{I}_K$ is the identity matrix, and $\mathbf{1}_K$ is the all-ones column vector. An ordinary token either retains its identity or becomes m; masked tokens remain masked throughout the forward process.

Alternative transition kernels. Uniform corruption mixes token retention with uniform resampling over ordinary codes, whereas embedding-based corruption favors transitions between codes with similar embeddings (Austin et al., 2021). Unlike absorbing masking, both permit substitutions between ordinary tokens. We compare these kernels in Table 1 under a fixed tokenization.

Conditioning and numerical control. For controllable generation, existing work steers only coarse attributes: ControLayout (Wang et al., 2024a) controls pattern category and topological complexity, while SCALE (Ho et al., 2026) uses natural-language rule descriptions to guide violation generation. However, neither method explicitly addresses numerical control of a critical dimension at a designated site. We address this setting through conditional token completion, providing a common framework for context-based rule-clean generation and numerically controlled rule-targeted generation.

2.3 DIVERSITY

We measure layout library diversity using topological-complexity entropy (Yang et al., 2022; Zhou et al., 2025b):

$$H_1(\mathcal{L}) = -\sum_{i,j} P_{\mathcal{L}}(i,j) \log_2 P_{\mathcal{L}}(i,j), \tag{3}$$

where $P_{\mathcal{L}}(i,j)$ is the fraction of patterns in library $\mathcal{L}$ with squish complexity $(C_x, C_y) = (i,j)$. Each component equals the number of distinct boundary-aligned scan lines along its axis, minus one.

2.4 PROBLEM FORMULATION

Following PatternPaint (Zhou et al., 2025b), we consider single-layer layout inpainting. Given a binary layout $x \in \{0,1\}^{H\times W}$ at fixed physical resolution and editable pixels $\mathcal{M}$, we seek diverse completions $\tilde{x} \in \{0,1\}^{H\times W}$ conditioned on $x_{\overline{\mathcal{M}}}$ and satisfying

$$\tilde{x}_{\overline{\mathcal{M}}} = x_{\overline{\mathcal{M}}}. \tag{4}$$

Completions need not recover the original masked content; they must be diverse and legal.

Problem 1: Rule-clean generation. Let $C_{\mathcal{R}}(x) \in \{0,1\}$ indicate whether x passes sign-off checking against the complete rule set $\mathcal{R}$. We seek completions satisfying $C_{\mathcal{R}}(\tilde{x}) = 1$.

Problem 2: Rule-targeted generation. For rule r, the seed instantiates a requested critical dimension $d^\star$ inside $\mathcal{M}$, e.g. a tip-to-side spacing of $18\,\mathrm{nm}$. A completion is *target-consistent* if some configuration of type r in $\mathcal{M}$ realizes $d^\star$ exactly, and *valid* if it further adds no violations of $\mathcal{R}$ beyond the seed, since a requested dimension may itself violate a nominal rule.

3 CODEPAINT

CodePaint combines a learned discrete layout vocabulary with absorbing diffusion to generate layout completions. The tokenizer captures local geometric features, while the diffusion model learns to complete their spatial configurations. Building on this representation, *TokenProbe* enables rule-targeted generation by probing encoder responses to geometric perturbations and selecting tokens to retain during completion, without additional training. Finally, we adapt the pretrained layout prior to new PDKs using limited target data. We detail these four components below.

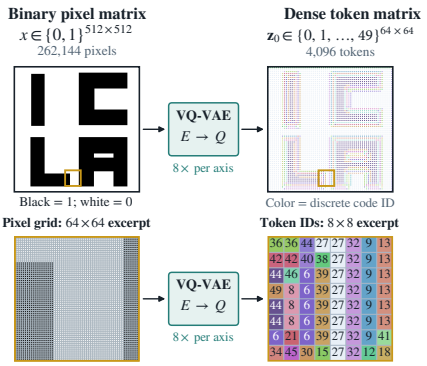


Figure 1: **VQ-VAE token space.** The encoder maps a binary 512×512 layout to a 64×64 grid of 64-dimensional features.

3.1 LEARNING A DISCRETE LAYOUT VOCABULARY

Binary layout rasters contain substantial spatial redundancy, making full-resolution pixel grids inefficient for generative modeling. Compressing a layout into a lower-resolution latent grid reduces the number of spatial locations that the generator must model. Moreover, IC layouts are composed of recurring local geometric patterns constrained by design rules, motivating their representation with a learned vocabulary of discrete codes. We therefore use a VQ-VAE to learn a compact discrete representation of layout geometry (Fig. 1). Following Eq. 1, an encoder E maps each layout x to local feature vectors, which are quantized using a learned codebook to produce a token grid $\mathbf{z}_0 = \mathcal{T}(x)$. A decoder D reconstructs the layout from the corresponding codebook embeddings (van den Oord et al., 2017). The diffusion model then learns the spatial configurations of these discrete layout features.

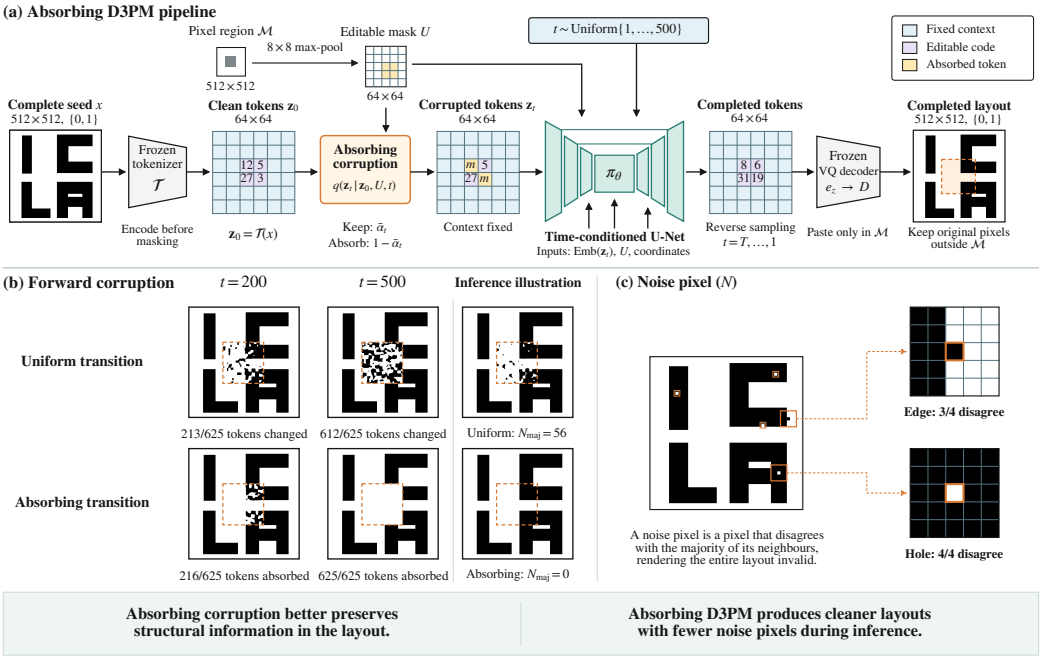


Figure 2: **Absorbing diffusion for conditional layout completion.** (a) Tokenization, completion, and decoding with a frozen VQ-VAE while preserving the original context. (b) Uniform and absorbing corruption within the editable region at $t = 200, 500$. (c) Schematic boundary defects and internal pinholes with local enlargements. N_{maj} counts editable-region pixels that differ from at least three of their four cardinal neighbors.

The encoder in our tokenizer maps each 512×512 layout to a 64×64 grid of 64-dimensional feature vectors (Fig. 1). We train the tokenizer with binary cross-entropy reconstruction, codebook, and commitment losses, using straight-through gradients through quantization. The encoder, codebook, and decoder are then frozen while the diffusion model learns a distribution over the spatial configurations of the resulting tokens. Architectural, vocabulary, and training details are provided in Appendix C. The VQ-VAE thus establishes a compact discrete representation of layout geometry. The remaining challenge is to learn a generative prior in this space that produces rule-compliant completions while preserving the unedited layout context.

3.2 Conditional Completion by Absorbing Diffusion

We exploit the discrete layout representation using an absorbing diffusion process that formulates generation as iterative token completion. We argue that absorbing diffusion is particularly well suited to layout generation because its information-removing corruption and monotonic denoising preserve structural context without introducing spurious layout tokens. Within the editable region, each token either retains its identity or transitions to a dedicated mask state m, while tokens outside the region remain fixed (Austin et al., 2021). Unlike uniform transitions, absorbing corruption never substitutes one ordinary layout code for another, thereby preserving the retained layout structure (Fig. 2b). Combined with monotonic completion, this process produces cleaner boundaries with fewer noise pixels (Fig. 2c). This is especially critical for DRC compliance, as a single pixel-scale defect can invalidate an otherwise clean layout.

Masked-token learning. We encode the seed layout into $\mathbf{z}_0$ and derive the editable token set $\mathcal{U}$ with binary indicator U from the pixel region $\mathcal{M}$. The transition in Eq. 2 applies only to $\mathcal{U}$; context tokens remain fixed. A time-conditioned denoiser predicts clean-code probabilities $\pi_{\theta,i}(k \mid \mathbf{z}_t, U, t)$ for $k \in \mathcal{V}$. We train it using cross-entropy over the masked editable positions $\mathcal{A}$:

$$\mathcal{L}_{\text{abs}} = -\frac{1}{|\mathcal{A}|} \sum_{(b,i)\in\mathcal{A}} \log \pi_{\theta,i}(z_{0,bi} \mid \mathbf{z}_{t_b,b}, U_b, t_b), \tag{5}$$

where b indexes samples and t_b is their sampled timestep.

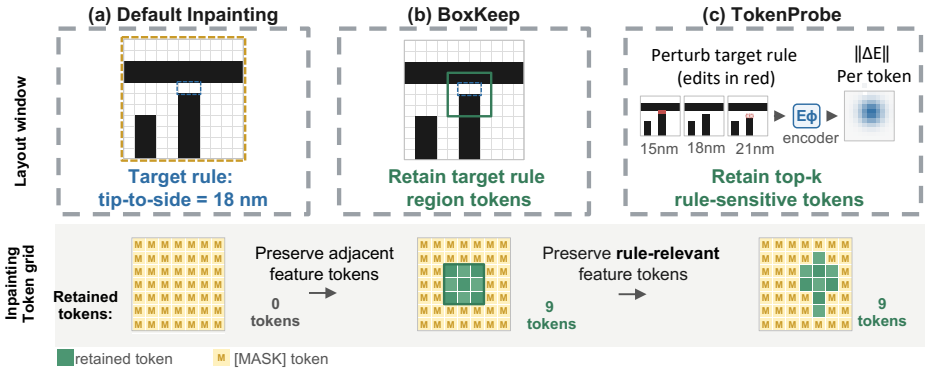


Figure 3: Token retention for an 18 nm tip-to-side spacing target. **(a)** Default inpainting masks all tokens in the neighborhood. **(b)** The BoxKeep baseline retains a fixed block around the target. **(c)** TokenProbe perturbs the spacing, measures encoder-feature changes, and retains the top-k sensitive tokens.

Progressive completion. Sampling initializes editable tokens to m while retaining the seed context (Fig. 2a). The seed is encoded with $\mathcal{M}$ zeroed out, so context tokens carry no source geometry from the editable region (Appendix C.2). With $\bar{\alpha}_t = \prod_{u=1}^{t}(1-\beta_u)$ and $\bar{\alpha}_0 = 1$, each masked token is revealed at reverse step t with probability

$$\eta_t = \frac{\bar{\alpha}_{t-1} - \bar{\alpha}_t}{1 - \bar{\alpha}_t}, \tag{6}$$

and receives a code sampled from $\pi_{\theta,i}$. Once revealed, the token remains fixed and becomes context for subsequent predictions (Austin et al., 2021). Finally, we decode the completed grid and replace only geometry inside $\mathcal{M}$, preserving the surrounding source layout. Although this full-deck prior achieves a high direct DRC-clean rate (Table 1), it does not reveal how information associated with an individual rule is encoded, making rule-specific control challenging.

3.3 TokenProbe for Rule-Targeted Generation

Rule-targeted generation probes a specific design rule by synthesizing layouts at requested geometric values, including violation-inducing or near-boundary values that may be rare or absent in the DRC-clean training distribution. The challenge is to impose such a target while retaining the learned layout prior for generating a plausible surrounding context. DeePattern (Yang et al., 2022) showed that perturbing individual latent dimensions can induce transformations in specific layout-rule domains, suggesting that learned representations encode rule-relevant geometric sensitivities. Building on this insight, we hypothesize that rule-specific geometry is carried by spatial tokens whose encoder features are sensitive to perturbations of the target dimension. Inspired by editing-mask generation in natural images (Couairon et al., 2023), we introduce *TokenProbe* to automatically identify and retain these geometry-sensitive tokens, thereby addressing Problem 2 in Sec. 2.4 (Fig. 3).

Token-retention setup. Let $x^{(r)}$ be a seed layout edited to realize the requested dimension r, and let $\mathbf{z}^{\text{seed}} = \mathcal{T}(x^{(r)})$. Within a token neighborhood B around the target site, we select a set $C \subseteq B$ of seed tokens to retain and regenerate the remaining tokens. Setting $C = \varnothing$ recovers default completion, which masks the entire neighborhood (Fig. 3a). As a baseline, *BoxKeep* retains a fixed block around the target site (Fig. 3b). Because encoder receptive fields overlap, spatial proximity alone may retain unnecessary context while missing tokens sensitive to the target dimension.

Selecting tokens by geometric sensitivity. TokenProbe selects tokens by their encoder-feature sensitivity to the target dimension (Fig. 3c). We perturb r, re-encode the edited layouts, and score each location by its maximum encoder-feature change:

$$s_i = \max_{\delta \in \Delta} \left\| E(x^{(r+\delta)})_i - E(x^{(r)})_i \right\|_2, \tag{7}$$

where Δ collects small signed perturbations of the target dimension (Sec. 4.3). For a retention budget k, we retain the highest-scoring tokens, $C_k = \text{top-k}_{i \in B}\, s_i$. Pre-quantization features capture sensitivity even when perturbations leave discrete token identities unchanged. The selection adapts to each layout and requires only encoder evaluations, without additional training.

Conditioned completion. We retain the seed-token identities in C_k, mask the remaining tokens in $B \setminus C_k$, and keep all tokens outside B fixed. We then apply the reverse process in Sec. 3.2 to complete the masked tokens. The retained tokens convey the requested geometry, while the masked tokens allow the learned layout prior to generate a compatible surrounding neighborhood.

3.4 Cross-PDK Adaptation of the Layout Prior

While TokenProbe enables rule-targeted generation of rare or unseen geometric values within the source PDK, adapting the pretrained layout prior to a new PDK with limited target data remains a complementary challenge. This setting is particularly important in early-stage DTCO, where too few routed layouts are available to train an effective generative model from scratch. Prior work PatternPaint (Zhou et al., 2025b) addresses this data scarcity by adapting a pretrained natural-image diffusion model for layout generation. We advance this workflow by adapting a prior pretrained directly on layout geometry to a new PDK using only N target clips.

We keep the encoder, codebook, and active vocabulary fixed to preserve the learned discrete token representation. To accommodate target-PDK geometry, we fine-tune the decoder using binary reconstruction loss on target layouts, together with a source-preservation term inspired by DreamBooth (Ruiz et al., 2023) that penalizes deviations from the frozen decoder on source-PDK token grids. We augment each target clip with four mirror orientations and eight sub-token pixel shifts, then re-encode each variant using the frozen encoder to cover different layout orientations and token-grid alignments.

To prevent frequent topologies from dominating adaptation, we combine the augmented target set $\mathcal{D}_{\text{tgt}}$ with a topology-balanced set $\mathcal{D}_{\text{tgt}}^{\text{bal}}$:

$$\mathcal{L}_{\text{adapt}} = \mathcal{L}_{\text{abs}}(\mathcal{D}_{\text{tgt}}) + \lambda\, \mathcal{L}_{\text{abs}}(\mathcal{D}_{\text{tgt}}^{\text{bal}}), \tag{8}$$

where $\mathcal{L}_{\text{abs}}$ is the masked-token objective in Eq. 5 and $\lambda = 1$. Both sets are derived from the same N target clips, and their losses are computed using separately sampled minibatches. At inference, we use the adapted denoiser and decoder with the frozen encoder and codebook.

4 Experiments

4.1 Experimental Setup

We extract 1,000 DRC-clean tiles from the M1 layer of a routed ASAP7 AES design and rasterize each tile as a 512×512 binary image with a pixel area of $1\,\text{nm}^2$. We partition the dataset by tile into 900 training and 100 test tiles and construct 11,700 and 1,300 layout–mask pairs, respectively. Each test tile is paired with 13 fixed masks, each covering approximately 25% of its area (Appendix B). Unlike prior evaluations based on three hand-coded rules (Yang et al., 2022; Wang et al., 2023), we stitch each completion back into its GDSII context and verify it with Calibre (Mentor Graphics, 2018) against all single-layer M1 checks in the ASAP7 deck, which includes spacing rules with thresholds that depend on the lengths of facing edges (Fig. 4, Appendix A).

We compare CodePaint with three continuous baselines: Pixel DDPM (Ho et al., 2020), operating directly in raster space; KL-LDM (Rombach et al., 2022), operating in the latent space of a KL-VAE trained on the same tiles; and PatternPaint (Zhou et al., 2025b), based on a fine-tuned SD2-inpaint model. We further conduct two groups of discrete ablations. To examine the effect of the corruption kernel, we compare absorbing, uniform, and codebook-similarity transitions (Austin et al., 2021) using the same VQ-VAE tokenizer (van den Oord et al., 2017). To examine the effect of tokenization, we replace VQ-VAE with FSQ (Mentzer et al., 2024) or Gumbel–Softmax (Jang et al., 2017), while retaining the absorbing transition kernel.

Clean denotes the fraction of the 1,300 test completions with zero Calibre (Mentor Graphics, 2018) violations. N_{maj} counts generated pixels that disagree with at least three of their four cardinal neighbors, aggregated across all completions, as a proxy for boundary speckle. H_1 (Eq. 3) denotes topological entropy, measured in bits over the deduplicated set of DRC-clean completions. We report metrics under two evaluation settings: *Direct*, which evaluates decoded layouts without geometric post-processing, and *+Denoise*, which applies the template-based cleanup of PatternPaint before evaluation. Model architectures, schedules, and hardware configurations are detailed in Appendix C.

Table 1: ASAP7 single-layer M1 DRC results, 1,300 layout–mask pairs per row. *Clean*: zero Calibre (Mentor Graphics, 2018) violations, before (*Direct*) and after (+*Denoise*) template cleanup. Shaded blocks vary only the transition kernel (upper) and only the tokenizer (lower). N_{maj} counts boundary-disagreement pixels and H_1 is entropy in bits, with the best value per column in bold. Cons.: Continuous, Disc.: Discrete.

Method	Formulation				Direct		+Denoise	
	Space	Variable	Transition	Tokenizer	Clean (%) ↑	N_{maj} ↓	Clean (%) ↑	H_1 ↑
Pixel DDPM (Ho et al., 2020)	Pixel	Cont.	Gaussian	–	3.23	**31**	11.62	6.51
KL-LDM (Rombach et al., 2022)	Latent	Cont.	Gaussian	KL-reg VAE	2.15	109,873	11.31	6.38
PatternPaint (Zhou et al., 2025b)	Latent	Cont.	Gaussian	KL-reg VAE	12.00	3,726	42.62	7.19
D3PM (Austin et al., 2021)	Latent	Disc.	Uniform	VQ-VAE	15.08	5,121	48.00	6.98
D3PM (Austin et al., 2021)	Latent	Disc.	Codebook-sim.	VQ-VAE	45.62	1,424	72.92	7.16
CodePaint	Latent	Disc.	Absorbing	FSQ	63.92	586	74.46	7.25
CodePaint	Latent	Disc.	Absorbing	Gumbel–Softmax	60.23	443	74.46	7.18
CodePaint	Latent	Disc.	Absorbing	VQ-VAE	**75.69**	206	**78.38**	**7.32**

4.2 Main Results

We ask three questions in turn: which generative *representation* suits layouts, which *corruption kernel* suits geometric constraint, and which *tokenizer* should define the discrete vocabulary (Table 1). All rows use the same 1,300-pair protocol and are evaluated with Calibre (Mentor Graphics, 2018) against all single-layer M1 checks.

Representation. Table 1 shows that discrete diffusion substantially outperforms continuous diffusion over pixels or latent features. CodePaint achieves 75.69% direct DRC compliance, 6.31× the best continuous baseline, and 78.38% after cleanup, at the highest diversity (H_1=7.32). PatternPaint reaches 12.00% directly and 42.62% only with cleanup, and the generic Pixel DDPM and KL-LDM, trained from scratch, are 3.23% and 2.15% clean directly and stay below 12% even after cleanup.

Corruption kernel. With the VQ-VAE fixed, the kernel alone moves the direct clean rate from 15.08% (uniform) to 45.62% (codebook-similarity) to 75.69% (absorbing); absorbing without cleanup already beats codebook-similarity with cleanup (72.92%). Meanwhile, absorbing state substantially reduces the number of boundary-disagreement pixels N_{maj} (206) relative to uniform (5,121) and codebook-similarity (1,424), which explains the improved geometric validity.

Tokenizer. With the absorbing kernel fixed, VQ-VAE beats FSQ and categorical (Gumbel-Softmax) tokenizers by 11.77 to 15.46 percentage direct and 3.92 after cleanup, so the gain is not from discretization alone. VQ tokens are nearest learned prototypes, so every code decodes to a local geometry observed in real layouts, whereas FSQ's fixed scalar grid and Gumbel-Softmax's logit-based assignment do not tie tokens to data-derived prototypes and produce more boundary speckle (N_{maj} 586 and 443 vs. 206).

Computational efficiency. CodePaint uses 54.85M parameters and achieves 1.891 images/s with 1.34 GiB peak memory, giving 7.13× higher throughput and 13.9× lower memory than SD2-inpaint (Appendix F.1).

4.3 Rule-Targeted Generation

Table 2: Rule-targeted generation on ASAP7 `M1.S.2`, pooled over six tip-to-side targets (15–28 nm, 3,600 draws per row); best rate in each column is bold. H_1 (Eq. 3, bits) is taken over each arm's own de-duplicated library of valid layouts.

Method	Target realization (%)			H_1 ↑
	Kept ↑	+ clean ↑	+ new ↑	
SCALE (Ho et al., 2026)	9.7	0.1	0.1	2.00
CodePaint, unconditional	10.0	3.7	2.2	6.08
CodePaint + Random	13.1	5.7	3.3	6.42
CodePaint + BoxKeep	38.9	14.7	8.8	7.01
CodePaint + TokenProbe	**55.2**	**22.5**	**12.4**	7.35

CodePaint supports inference-time control of local design-rule geometry without retraining. We evaluate tip-to-side spacing (M1.S.2, Fig. 4 and Appendix A) at six targets, 15, 18, 20, 22,

25, and 28 nm, where 25 nm is the minimum spacing allowed by the rule. TokenProbe uses $\Delta = \{\pm 1, \pm 2, \pm 3\}$ nm in Eq. 7.

For each target, we modify one site per ASAP7 clip and regenerate a token neighborhood B around it while keeping every token outside B fixed (full protocol and per-target breakdown in Appendix D). Unconditional sampling retains no tokens inside B, while the three retention methods each retain exactly nine. Random draws them anywhere in B, BoxKeep takes the 3×3 block centered on the site, and TokenProbe takes the nine highest-scoring tokens under Eq. 7. We also compare against our reimplementation of SCALE (Ho et al., 2026) (Appendix D.2), a recent autoregressive layout generator conditioned on text prompts. A draw is *Kept* if it contains a tip-to-side pair at exactly the requested spacing that overlaps B. + *clean* further requires no new Calibre (Mentor Graphics, 2018) violations after cleanup and stitching, relative to the seed reference. + *new* additionally requires the regenerated window to differ from that reference. We also report the diversity H_1 (Eq. 3) over each method's library of valid layouts.

TokenProbe improves valid target realization. At the same token-retained budget, TokenProbe raises *Kept* + *clean* from 14.7% with BoxKeep to 22.5%, compared with 3.7% without retained tokens, 5.7% with Random, and 0.1% with the text-conditioned SCALE baseline (Table 2). It also raises + *new* from 8.8% with BoxKeep to 12.4%, compared with 2.2% unconditionally and 3.3% with Random. TokenProbe also attains the highest diversity among all methods ($H_1 = 7.35$ bits vs. 7.01 for BoxKeep). These results demonstrate that selective token retention delivers more effective target control without sacrificing generation diversity.

4.4 Few-Shot Commercial PDK Adaptation

We evaluate ASAP7-to-commercial-16 nm transfer by fine-tuning CodePaint and PatternPaint on 20, 50, or 100 target-PDK clips (full setting in Appendix E). Table 3 reports compliance with the target Calibre (Mentor Graphics, 2018) deck before and after template cleanup, and topological entropy H_1 over the distinct DRC-clean denoised outputs.

Table 3: Few-shot adaptation to a commercial 16 nm PDK from the ASAP7 model (CodePaint) or the public SD2-inpaint checkpoint (PatternPaint), where zero-shot denotes inference with these base models without fine-tuning. Direct and +Denoise are DRC-clean rates before and after cleanup, and H_1 is computed over de-duplicated clean outputs; row-best values are bold.

Fine-tuning clips	PatternPaint			CodePaint (ours)		
	Direct (%) ↑	+Denoise (%) ↑	H_1 ↑	Direct (%) ↑	+Denoise (%) ↑	H_1 ↑
zero-shot	**2.54**	**3.15**	**2.762**	1.85	1.92	0.956
20	1.69	5.31	3.218	**27.54**	**29.31**	**3.597**
50	2.31	6.62	3.229	**35.69**	**35.85**	**3.848**
100	4.69	10.38	3.671	**40.62**	**41.08**	**4.012**

Strong adaptation with limited target-PDK data. With just 20 clips, CodePaint achieves 29.3% post-cleanup compliance, exceeding PatternPaint with 100 clips (10.38%). At 100 clips, CodePaint reaches 40.6% direct and 41.1% post-cleanup compliance. Diversity also increases with target data (H_1: 3.60 to 4.01) and exceeds PatternPaint at every clip count. These results show that a prior pretrained on layout geometry transfers to a commercial node with far fewer target examples than a prior pretrained on natural images: CodePaint with 20 clips exceeds PatternPaint with 100.

5 Conclusion

We presented CodePaint, a framework that combines a learned discrete layout vocabulary with absorbing diffusion for valid and controllable layout generation. Generating configurations of local geometric features, rather than pixels or continuous latents, raises direct DRC compliance by over $6\times$ relative to continuous baselines and makes the token grid an interface for training-free geometric control, which TokenProbe exploits to hit requested critical dimensions without retraining. The same prior transfers to a commercial 16 nm PDK from 20 target clips, outperforming PatternPaint given 100. Together, these results show that a prior learned from layout geometry can support rule-clean generation, targeted rule exploration, and sample-efficient transfer within a shared framework, providing a practical basis for pattern-library construction in early-stage DTCO.

Ethics Statement

This work involves no human subjects, no personally identifiable information, and no crowdsourced annotation. The pretraining corpus is derived from the open-source ASAP7 predictive PDK (Clark et al., 2016) and the designs routed with it, which carry no confidentiality restrictions. We report all results against a commercial physical-verification tool rather than a reimplementation, so that the reported compliance rates reflect sign-off checking rather than a proxy of our own construction. We see no foreseeable harm from this work beyond the ordinary dual-use considerations of design automation research.

Reproducibility Statement

We describe the design rules used for evaluation in Appendix A and our dataset construction in Appendix B. Model architectures, training objectives, hyperparameters, and the sampling and physical-verification protocol are given in Appendix C, and the few-shot adaptation recipe in Appendix E. We provide open-source code through an anonymized repository[1], including tokenizer and diffusion training, token retention, few-shot adaptation, and the evaluation scripts, together with our ASAP7-derived dataset.

The Use of Large Language Models (LLMs)

LLM-based assistants were used only for writing support, namely for language editing of author-written text and for routine coding assistance such as plotting and table generation scripts. They were not used to generate research ideas, to design or run experiments, to analyze results, or to produce any of the reported numbers, and no component of CodePaint uses an LLM. The authors verified every LLM-assisted edit and take full responsibility for the content of this paper.

References

ASAP7. *ASAP7 PDK Design Rule Manual*. Arizona State University, pdk release 1p7 (rev. 201207a) edition, 2020. URL https://github.com/The-OpenROAD-Project/asap7_pdk_r1p7.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 17981–17993, 2021.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. MaskGIT: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Ran Chen, Wei Zhong, Haoyu Yang, Hao Geng, Xuan Zeng, and Bei Yu. Faster region-based hotspot detection. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2019. doi: 10.1145/3316781.3317824.

Lawrence T. Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. ASAP7: A 7-nm FinFET predictive process design kit. *Microelectronics Journal*, 53:105–115, 2016.

Guillaume Couairon, Jakob Verbeek, Holger Schwenk, and Matthieu Cord. DiffEdit: Diffusion-based semantic image editing with mask guidance. In *International Conference on Learning Representations (ICLR)*, 2023.

Jhih-Rong Gao, Xiaoqing Xu, Bei Yu, and David Z. Pan. MOSAIC: Mask optimizing solution with process window aware inverse correction. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2014.

[1] https://anonymous.4open.science/r/CODEPAINT_ICLR_RELEASE-92F0/

Elizabeth Grubbs, Enas Sakr, Yan Feng, Jirka Schatz, Parag Ghosh, Lena Zavyalova, Folarin Latinwo, Zac Levinson, Juergen Preuninger, and Kevin Lucas. Beyond design-rule test pattern coverage for low-K1 hotspot and process variability reduction. In *DTCO and Computational Patterning IV*, volume 13425 of *Proceedings of SPIE*, pp. 134250Z, 2025. doi: 10.1117/12.3055695.

Shuyang Gu, Dong Chen, Jianmin Bao, Fang Wen, Bo Zhang, Dongdong Chen, Lu Yuan, and Baining Guo. Vector quantized diffusion model for text-to-image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10686–10696, 2022. doi: 10.1109/CVPR52688.2022.01043.

Ayman Hamouda, Mohamed Bahnas, Dan Schumacher, Ioana Graur, Ao Chen, Kareem Madkour, Hussein Ali, Jason Meiring, Neal Lafferty, and Chris McGinty. Enhanced OPC recipe coverage and early hotspot detection through automated layout generation and analysis. In *Optical Microlithography XXX*, volume 10147, pp. 101470R. SPIE, 2017. doi: 10.1117/12.2260769.

Chia-Tung Ho, Haoyu Yang, Guanglei Zhou, Yoshi Nishi, Yaguang Li, Walker Turner, Cunxi Yu, Yiran Chen, and Brucek Khailany. SCALE: Self-supervised constraint-aware layout GEneration for local P&R DRV fixing at advanced nodes, 2026. URL https://arxiv.org/abs/2607.21850.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 6840–6851, 2020.

Naoto Inoue, Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. LayoutDM: Discrete diffusion model for controllable layout generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10167–10176, 2023. doi: 10.1109/CVPR52729.2023.00980.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

Bentian Jiang, Hang Zhang, Jinglei Yang, and Evangeline F. Y. Young. A fast machine learning-based mask printability predictor for OPC acceleration. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 412–419, 2019. doi: 10.1145/3287624.3287682.

Yiyang Jiang, Fan Yang, Bei Yu, Dian Zhou, and Xuan Zeng. Efficient layout hotspot detection via neural architecture search. *ACM Transactions on Design Automation of Electronic Systems*, 27 (6):62:1–62:16, 2022. doi: 10.1145/3517130.

I. T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, New York, second edition, 2002. doi: 10.1007/b98835.

Aliaa Kabeel, Sutae Kim, Young Gook Park, Donggyun Kim, Joe Kwan, Sarah Rizk, Kareem Madkour, Marwa Shafee, and Jinhee Kim. Design rule manual and DRC code qualification flows empowered by high coverage synthetic layouts generation. In *DTCO and Computational Patterning II*, volume 12495 of *Proceedings of SPIE*, pp. 124951T, 2023. doi: 10.1117/12.2658350.

Abde Ali Kagalwalla, Swamy Muddu, Luigi Capodieci, Coby Zelnik, and Puneet Gupta. Design-of-experiments based design rule optimization. In *Design for Manufacturability through Design-Process Integration VI*, volume 8327 of *Proceedings of SPIE*, pp. 832709, 2012. doi: 10.1117/12.918067.

Mentor Graphics. *Calibre Verification User's Manual*. Mentor Graphics, 2018.

Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: Vq-vae made simple. In *International Conference on Learning Representations*, 2024.

Gaurav Rajavendra Reddy, Kareem Madkour, and Yiorgos Makris. Machine learning-based hotspot detection: Fallacies, pitfalls and marching orders. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, 2022.

Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. DreamBooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Synopsys. What is design rule checking (DRC)? Synopsys website. URL https://www.synopsys.com/glossary/what-is-design-rule-checking.html. Accessed September 22, 2026.

The OpenROAD Project. OpenROAD-flow-scripts: AES benchmark source documentation. GitHub repository documentation. URL https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/blob/master/flow/designs/src/aes/README.md. Accessed: 2026-09-21.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

Qijing Wang, Xiaopeng Zhang, Martin D.F. Wong, and Evangeline F.Y. Young. ControLayout: Conditional diffusion for style-controllable and violation-fixable layout pattern generation. In *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 511–515, 2024a. doi: 10.1145/3649476.3658770.

Zixiao Wang, Yunheng Shen, Wenqian Zhao, Yang Bai, Guojin Chen, Farzan Farnia, and Bei Yu. DiffPattern: Layout pattern generation via discrete diffusion. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2023. doi: 10.1109/DAC56929.2023.10248009.

Zixiao Wang, Yunheng Shen, Xufeng Yao, Wenqian Zhao, Yang Bai, Farzan Farnia, and Bei Yu. ChatPattern: Layout pattern customization via natural language. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 2024b.

Zixiao Wang, Wenqian Zhao, Yunheng Shen, Yang Bai, Guojin Chen, Farzan Farnia, and Bei Yu. DiffPattern-Flex: Efficient layout pattern generation via discrete diffusion. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 44(12):4679–4690, 2025. doi: 10.1109/TCAD.2025.3569493.

Xiang Xu, Karl D. D. Willis, Joseph G. Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. SkexGen: Autoregressive generation of CAD construction sequences with disentangled codebooks. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pp. 24698–24724. PMLR, 2022.

Xiang Xu, Joseph G. Lambourne, Pradeep Kumar Jayaraman, Zhengqing Wang, Karl D. D. Willis, and Yasutaka Furukawa. BrepGen: A B-rep generative diffusion model with structured latent geometry. *ACM Transactions on Graphics*, 43(4):119:1–119:14, 2024. doi: 10.1145/3658129.

Haoyu Yang, Shuhe Li, Wen Chen, Piyush Pathak, Frank Gennari, Ya-Chieh Lai, and Bei Yu. DeePattern: Layout pattern generation with transforming convolutional auto-encoder. *IEEE Transactions on Semiconductor Manufacturing*, 35(1):67–77, 2022. doi: 10.1109/TSM.2021.3139354.

Ye Yuan, Weien Li, Rui Song, Zeyu Li, Haochen Liu, Xiangyu Kong, Zixuan Dong, Linfeng Du, Zipeng Sun, Weixu Zhang, Jiaxin Huang, Changjiang Han, Yonghan Yang, Zichen Zhao, Xiuyuan Hu, Haolun Wu, Yankai Chen, Fengran Mo, Jikun Kang, Bowei He, Dawn Song, Philip S. Yu, and Xue Liu. Discrete diffusion models: A unified framework from tokenization to generation, 2026. URL https://arxiv.org/abs/2607.13431.

Junyi Zhang, Jiaqi Guo, Shizhao Sun, Jian-Guang Lou, and Dongmei Zhang. LayoutDiffusion: Improving graphic layout generation by discrete diffusion probabilistic models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7192–7202, 2023. doi: 10.1109/ICCV51070.2023.00664.

Xiaopeng Zhang, James Shiely, and Evangeline F.Y. Young. Layout pattern generation and legalization with generative learning models. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–9, 2020.

Xiaopeng Zhang, Haoyu Yang, and Evangeline F.Y. Young. Attentional transfer is all you need: Technology-aware layout pattern generation. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pp. 169–174, 2021.

Tianxiong Zhong, Xingye Tian, Xuebo Wang, Xin Tao, and Pengfei Wan. Diffusing in the right space: A systematic study of latent diffusability, 2026. URL https://arxiv.org/abs/2606.03578.

Guanglei Zhou, Chen-Chia Chang, Junyao Zhang, Jingyu Pan, and Yiran Chen. Diffusion-model-enhanced layout pattern generation for sub-3nm DFM. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2025a. doi: 10.1109/ICCAD66269.2025.11240639. URL https://jpan.me/files/ICCAD25_Diffusion_DFM.pdf.

Guanglei Zhou, Bhargav Korrapati, Gaurav Rajavendra Reddy, Chen-Chia Chang, Jingyu Pan, Jiang Hu, Yiran Chen, and Dipto G. Thakurta. PatternPaint: Practical layout pattern generation using diffusion-based inpainting. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7, 2025b. doi: 10.1109/DAC63849.2025.11132857.

Guanglei Zhou, Bhargav Korrapati, Gaurav Rajavendra Reddy, Junyao Zhang, Yiran Chen, and Dipto G. Thakurta. Vario: Enhance pattern diversity with diffusion model. In *DTCO and Computational Patterning IV*, volume 13425 of *Proceedings of SPIE*, 2025c. doi: 10.1117/12.3049792.

CONTENTS OF THE APPENDIX

A ASAP7 M1 Design Rules 15
B The 1,000-Tile AES Layout Dataset 16
Source design and layout preparation 16
Tiling and image representation 16
Diversity selection and the 900/100 partition 16
Inpainting mask protocols 17
C Implementation Details 19
Overall Architecture 19
Two-Stage Training 20
Diffusion Process and Training Objectives 20
Training Settings and Computational Cost 21
Baseline and Ablation Training Settings 21
Model Size and Sampling Cost 22
Sampling and Physical Verification 23
D Rule-Targeted Generation Details 24
Clips, Targets, and Regenerated Neighborhoods 24
Compared Methods 24
Cleanup, Design-Rule Checking, and Metrics 24
Results by Target 25
E Few-Shot Cross-PDK Adaptation: Experimental Setting 26
Target-PDK data 26
CodePaint adaptation 26
Evaluation 27
F Additional Results 28
Model Size and Inference Efficiency 28
DRC-clean layout completion 28
Sampling diversity 28
Pixel noise 31

A ASAP7 M1 Design Rules

Fig. 4 is our own illustration of the selected ASAP7 M1 rules that this work conditions on and evaluates against, and is not a complete specification. The full rule set, including the rules omitted here and the exact conditions under which each applies, is given in the ASAP7 design rule manual (Clark et al., 2016).

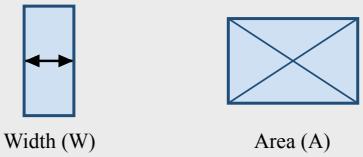
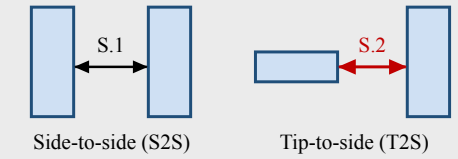
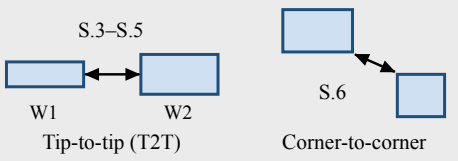
<table>
<tr><th>Layout Illustration</th><th>Rule Description (ASAP7 M1)</th></tr>
<tr><td colspan="2">Width and Area: M1.W.1, M1.A.1</td></tr>
<tr><td>
</td><td>M1.W.1 Minimum width: $W \geq 18$ nm
M1.A.1 Minimum area: $A \geq 504$ nm^2</td></tr>
<tr><td colspan="2">Spacing, Conditioned on Facing-Edge Length: M1.S.1–M1.S.6</td></tr>
<tr><td>
</td><td>M1.S.1 Side-to-side, both facing edges > 36 nm:
$S \geq 18$ nm
M1.S.2 Tip-to-side, one facing edge ≤ 36 nm:
$S \geq 25$ nm (conditioning target in this work)</td></tr>
<tr><td>
</td><td>M1.S.3–M1.S.5 Tip-to-tip, set by the two tip widths:
<table>
<tr><th>W2 \ W1</th><th>< 24 nm</th><th>24–36 nm</th></tr>
<tr><td>< 24 nm</td><td>S.4: ≥ 31 nm</td><td>S.5: ≥ 31 nm</td></tr>
<tr><td>24–36 nm</td><td>S.5: ≥ 31 nm</td><td>S.3: ≥ 27 nm</td></tr>
</table>
M1.S.6 Corner-to-corner: $S \geq 20$ nm</td></tr>
</table>

Figure 4: Single-layer ASAP7 M1 design rules used in this work. Every spacing rule is conditioned on the lengths of the two facing edges: a wire tip (≤ 36 nm) facing a long side falls under M1.S.2 (red), our conditioning target, and tip-to-tip spacing is set by both tip widths.

B THE 1,000-TILE AES LAYOUT DATASET

The dataset comprises 1,000 M1 layout tiles extracted from a single AES encryption design, with 900 tiles assigned to training and 100 to testing. We use the same source tiles for VQ-VAE and KL-VAE representation learning and for the inpainting experiments. Dataset construction consists of rasterizing candidate layout regions, selecting diverse tiles in a principal-component representation, and pairing the selected tiles with inpainting masks. Throughout this appendix, a *source tile* is a layout crop, whereas an *inpainting record* is a source tile paired with a mask. Additional masks or stochastic generations do not increase the number of source tiles.

B.1 SOURCE DESIGN AND LAYOUT PREPARATION

We obtain the AES RTL and technology collateral from OpenROAD-flow-scripts (ORFS), whose benchmark documentation attributes the design to the OpenCores AES core (The OpenROAD Project). The encryption top module is `aes_cipher_top`. We synthesize the RTL with Yosys and map it to ASAP7 regular-threshold-voltage (RVT) standard cells, then perform placement and routing in Cadence Innovus. Thus, ORFS provides the source materials rather than the physical implementation flow used here. The execution logs record Yosys 0.66+181 (revision `afe6b18f2`) and Innovus `21.17-s075_1`. The technology files include the ASAP7 7.5-track RVT cell library and its LEF, Liberty, and GDS views. ASAP7 is a predictive 7-nm FinFET process design kit (Clark et al., 2016).

We extract M1 from GDS layer/datatype `19/0` and normalize the coordinates to 4,000 database units (DBU) per micrometer without changing physical dimensions; one DBU therefore corresponds to 0.25 nm. Rule annotations are derived from Calibre (Mentor Graphics, 2018) checks on the source layout. The executed checks cover M1 width, spacing, and area rules: `M1.W.1`, `M1.S.1`–`M1.S.6`, and `M1.A.1` (ASAP7). The recorded results contain no violations of these checks, and all 1,000 selected tiles have zero annotated violations within this rule scope. Inpainting inputs are formed by masking geometry in these source tiles.

B.2 TILING AND IMAGE REPRESENTATION

We cover the M1 bounding box with a regular two-dimensional grid of square windows. Each window has a side length of 2,048 DBU (512 nm), with a stride of 1,792 DBU (448 nm) along each axis. The grid includes windows extending beyond the upper or right edge of the bounding box. Geometry is clipped to each window and rasterized into a 512×512 image at 4 DBU per pixel, or 1 nm per pixel. The selection procedure below retains 1,000 of these candidate tiles.

Rasterization determines occupancy at pixel centers. Image row zero corresponds to the upper edge of the window, so row indices increase in the direction opposite to the layout's vertical coordinate. The grayscale PNGs encode background as 0 and M1 as 191; a threshold of 127 converts these values to binary background and metal occupancy for the binary layout models. Source tiles are stored at 512×512 resolution, independently of any model-specific training crops. The 64×64 images described below are used to construct selection features. Each tile's metadata records its source ID, GDS bounding box in DBU and micrometers, geometry count, and links to its image and rule annotations.

B.3 DIVERSITY SELECTION AND THE 900/100 PARTITION

We select a compact, structurally diverse set of source tiles using greedy farthest-point sampling in a principal-component representation. PCA is fitted to the candidate descriptors before the selected tiles are partitioned into training and test sets.

Selection features. We first sort candidate records lexicographically by source ID. Each grayscale PNG is downsampled to 64×64 using BOX resampling, converted to floating point, divided by 255, and flattened into a 4,096-dimensional vector. This yields a coarse, intensity-scaled metal-density descriptor that retains the quantization introduced by image resampling. We apply principal component analysis (PCA) (Jolliffe, 2002) to these descriptors and retain the leading 64 components. The decomposition uses randomized SVD with seed 42 and four power iterations. These components

Table 4: Inpainting records derived from the same 900 training and 100 test source tiles. Each random-shape row denotes a separate missing-area protocol. Repeated stochastic generations are excluded.

Mask protocol	Masks/tile	Records	
		Training	Test
Fixed positions, $\approx 25\%$	13	11,700	1,300
Random shapes, 25%	12	10,800	1,200
Random shapes, 50%	12	10,800	1,200
Random shapes, 75%	12	10,800	1,200

explain approximately 54.96% of the feature variance. The unwhitened PCA scores, stored in single precision, serve as the sampling coordinates.

Greedy selection. Let $\mathbf{p}_i \in \mathbb{R}^{64}$ denote the PCA score of candidate i, and let S_k be the set of indices selected after k iterations. We initialize and expand the selected set as follows:

$$i_1 = \arg\max_i \|\mathbf{p}_i\|_2^2, \qquad S_1 = \{i_1\}, \tag{9}$$

$$i_{k+1} = \arg\max_{i \notin S_k} \min_{j \in S_k} \|\mathbf{p}_i - \mathbf{p}_j\|_2^2, \qquad S_{k+1} = S_k \cup \{i_{k+1}\}. \tag{10}$$

Selection stops when $|S_k| = 1{,}000$. Because PCA centers the descriptors, the initial sample is the candidate farthest from the origin in the centered score space. At each subsequent iteration, we update each remaining candidate's minimum distance to the selected set. Ties are resolved by the first maximum in the ordered candidate array. This procedure promotes diversity in the selection representation without using rule labels or model performance. The source manifest records each selected tile's selection rank and the distance used at the time of selection.

Source-level partition. We permute the 1,000 selection ranks using NumPy's `default_rng(42)`, assigning the first 900 to training and the remaining 100 to testing. The two partitions have disjoint source IDs. We additionally verified that no training and test tiles overlap in source-layout coordinates. Masks are generated after this assignment, and every record derived from a source tile inherits its partition. All fixed-position and random-shape mask protocols use these same 900 training and 100 test tiles. The 100-tile test partition is named `val` in the manifests.

B.4 Inpainting mask protocols

Let $x \in \{0,1\}^{512\times512}$ be a binary source layout and $M \in \{0,1\}^{512\times512}$ a pixel mask, where $M = 1$ denotes a missing pixel. The observed layout is

$$x_{\text{obs}} = (1 - M) \odot x. \tag{11}$$

The mask is stored separately to distinguish missing pixels from observed background. Mask PNGs encode missing pixels as white (255), and the data loaders set masked image pixels to zero. Every protocol preserves an unmasked boundary frame at least 32 pixels (32 nm) wide. Missing-area fractions are measured relative to the entire tile, not its metal area or the interior available for masking. Table 4 summarizes the resulting record counts.

Fixed-position masks. This protocol uses 13 templates shared by all source tiles: four inset corner squares, four vertical bands, four horizontal bands, and one central square. Each square measures 256×256 pixels and masks exactly 25% of the tile. In zero-based image coordinates (x, y), with half-open intervals, the corner-square origins are $(32, 32)$, $(224, 32)$, $(32, 224)$, and $(224, 224)$; the central square starts at $(128, 128)$. Each vertical band measures 146×448 pixels, spans $y \in [32, 480)$, and starts at $x \in \{32, 133, 233, 334\}$. Horizontal bands are the transposes of these vertical bands. Each band masks 65,408 pixels, or approximately 24.9512% of the tile. Applying all 13 templates to each source tile yields 11,700 training and 1,300 test records.

Random shapes with controlled missing area. For each target missing-area fraction $r \in \{0.25, 0.50, 0.75\}$, we generate three instances from each of four shape families per source tile: rectangle-like regions, elongated serpentine stripes, triangle-based regions, and irregular connected regions. Each fraction therefore yields 12 records per tile, or 10,800 training and 1,200 test records. Each shape family contributes 2,700 training and 300 test records at each fraction.

Masks are constructed on a 64×64 grid with a four-cell unmasked frame, leaving a 56×56 drawable interior. Each mask contains exactly $r \cdot 64^2$ missing cells: 1,024, 2,048, or 3,072 for the three fractions. Replicating each cell into an 8×8 pixel block produces exactly 65,536, 131,072, or 196,608 missing pixels. These masks are aligned to an eight-pixel grid, so 8×8 max-pooling with stride eight exactly recovers the construction mask. Generation enforces four-neighbor connectivity and the unmasked boundary frame. In contrast, fixed-position templates are defined directly in pixel coordinates and need not follow this grid alignment.

All four families use connected growth to reach the prescribed area:

- **Rectangle-like regions.** Growth follows a rotated rectangular priority field, with an angle sampled within $\pm 20°$, an aspect ratio sampled log-uniformly from $[0.78, 1.28]$, and a small beveling term.
- **Elongated stripes.** A serpentine centerline with three to six lanes is randomly rotated and reflected, then thickened according to distance from the centerline.
- **Triangle-based regions.** Barycentric-coordinate priority fields guide growth around a single triangle or the union of two triangles. The 25% protocol uses a single-triangle field; the 50% and 75% protocols use two-triangle fields to accommodate the larger masked areas within the inset region. The resulting masks are connected regions grown to the exact target area.
- **Irregular connected regions.** Growth begins at a random cell and follows a random field smoothed by three local averaging passes, with an additional distance penalty from the seed.

At 75% missing area, 3,072 of the 3,136 drawable interior cells are masked. All four families remain subject to the same exact-area and boundary constraints.

Deterministic generation and metadata. Random-mask generation uses master seed 1234. Each mask receives an unsigned 64-bit seed derived from the master seed, fraction tag, partition, source ID, shape family, and instance index. These values are serialized as a compact JSON list of strings and hashed with SHA-256; the first eight digest bytes are interpreted as a big-endian integer. A NumPy random generator initialized with this seed produces the mask, making generation independent of source-tile traversal order. For each source tile and fraction, duplicate grid masks are rejected across all four families and three instances, with retries continuing the same random stream. The manifests record seeds, mask areas, bounding boxes, source mappings, and hashes of the pixel PNGs and packed grid masks. Checksums of the source and derived manifests identify the dataset version used by each protocol.

C Implementation Details

This appendix describes the ASAP7 rule-clean experiments, including model configurations, training, and physical verification. Figure 5 summarizes the pipeline.

C.1 Overall Architecture

For a binary M1 raster $x \in \{0,1\}^{512\times512}$, one denotes metal. The editable pixel region $\mathcal{M}$ has binary mask M; its latent counterpart is $\mathcal{U} = \{i : U_i = 1\}$, with $U \in \{0,1\}^{64\times64}$. We denote the absorbing state by $m = [\texttt{MASK}]$.

Layout tokenizer. The VQ-VAE maps x to a 64×64 grid of 64-dimensional features, quantized by nearest-neighbor lookup in a 128-vector codebook. The 50 codes observed in the 900 training tiles define the active vocabulary $\mathcal{V}$, with $K = |\mathcal{V}| = 50$. They are remapped in ascending original-ID order to $0, \ldots, 49$; all test-tile codes belong to this vocabulary. The absorbing symbol has ID 50, giving 51 denoiser input symbols and 50 output classes. Decoding reverses this ID mapping.

The encoder uses three 4×4 stride-two convolutions with 64/128/256 channels, two residual blocks (each with 3×3 and 1×1 convolutions and ReLU), and a 1×1 projection to 64 channels. This stack has a theoretical 54×54-pixel receptive field and stride eight. The decoder projects embeddings to 256 channels, applies two residual blocks, and upsamples through three 4×4 stride-two transposed convolutions to single-channel occupancy logits.

Diffusion denoiser. The time-conditioned U-Net uses 128/256/512 channels at $64 \times 64/32 \times 32/16 \times 16$ resolutions, two residual blocks per encoder/decoder stage, and four-head attention at 16×16 and in the bottleneck. Residual blocks use GroupNorm, SiLU, dropout 0.1, and time-conditioned scale and shift. Sinusoidal timestep embeddings pass through an MLP with 512-dimensional output. Inputs concatenate learned 128-dimensional token embeddings, U, and two normalized coordinate channels; the output is the clean-code distribution $\pi_{\theta,i}(k \mid \mathbf{z}_t, U, t)$. The denoiser embeddings are distinct from the frozen VQ embeddings.

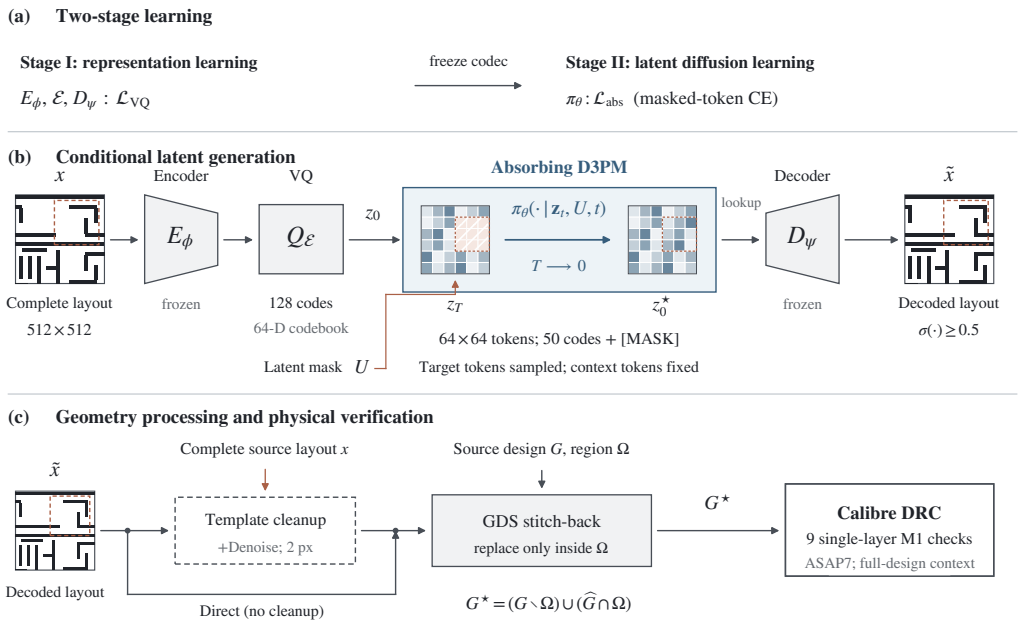


Figure 5: Two-stage training and generation pipeline. The VQ-VAE is trained first and frozen for diffusion training on cached tokens. Sampling completes editable tokens while fixing context. Direct omits geometric cleanup; +Denoise applies it. Both branches replace only geometry in the original pixel-mask region of the source GDSII layout before Calibre DRC (Mentor Graphics, 2018).

C.2 Two-Stage Training

Data and tokenizer training. Both stages use 900 training and 100 test source tiles. Each tile is paired with 13 fixed masks, yielding 11,700 training and 1,300 test pairs. Test tiles are used only for reported evaluation, and no checkpoint is chosen from test-tile or Calibre results. The dataset's random-shape protocols are not used here. VQ-VAE training alternates between native 256×256 crops in odd epochs and complete 512×512 tiles in even epochs. Crop origins lie on a two-pixel grid; no resizing is applied. We use the epoch-2,500 checkpoint, after the training BCE has plateaued. On the 100 test tiles, this tokenizer has BCE 1.6322×10^{-6}, a mean of 0.10 erroneous pixels per tile, and 94/100 exact reconstructions. It remains frozen during ASAP7 diffusion training and evaluation. Optimization settings are in Table 5.

Token masking and conditioning. Each complete source layout is encoded once as $\mathbf{z}_0 = \mathcal{T}(x)$ and cached, providing the clean target tokens for the training loss. We compute $U = \mathrm{MaxPool}_{8\times 8,\, s=8}(M)$, marking every token block that intersects $\mathcal{M}$. Nonaligned pixel masks can therefore produce editable token blocks extending outside $\mathcal{M}$. Only editable tokens are corrupted and regenerated, and context tokens remain. Because encoder receptive fields overlap, encoding a complete layout would let retained boundary tokens depend on source geometry inside the editable region. To avoid context tokens incorrectly carrying such masked source information, we zero out the pixels of $\mathcal{M}$ before encoding at generation time, so context tokens are computed from visible geometry only. We establish this as a general protocol for all model inference: no inference input contains source geometry inside the editable region.

C.3 Diffusion Process and Training Objectives

Tokenizer objective. For encoder features $\mathbf{h} = E(x)$ and assigned codebook embeddings $\mathbf{z}_q$, the tokenizer minimizes

$$\mathcal{L}_{\mathrm{VQ}} = \mathrm{BCE}(x, \sigma(D(\mathbf{z}_q))) + \|\,\mathrm{sg}(\mathbf{h}) - \mathbf{z}_q\|_2^2 + 0.25\|\mathbf{h} - \mathrm{sg}(\mathbf{z}_q)\|_2^2, \tag{12}$$

where σ is sigmoid and sg stops gradients. BCE is unweighted; both squared-error terms are elementwise means. Straight-through reconstruction gradients update the encoder, and gradient descent updates the codebook. No KL term is used.

Forward process. For editable tokens, the transition and cumulative survival probability are

$$\mathbf{Q}_t^{\mathrm{abs}} = \begin{bmatrix} (1-\beta_t)\mathbf{I}_K & \beta_t \mathbf{1}_K \\ \mathbf{0}_K^{\mathsf{T}} & 1 \end{bmatrix}, \qquad \bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s), \quad \bar{\alpha}_0 = 1. \tag{13}$$

Thus $q(z_{t,i} \mid z_{0,i}) = \bar{\alpha}_t \delta_{z_{0,i}} + (1-\bar{\alpha}_t)\delta_m$ for $U_i = 1$; context tokens follow the identity transition. With $T = 500$, the cosine schedule is

$$f(t) = \cos^2\left(\frac{t/T + s}{1+s}\frac{\pi}{2}\right), \qquad s = 0.008, \tag{14}$$

$$\beta_t = \mathrm{clip}\left(1 - \frac{f(t)}{f(t-1)}, 10^{-8}, 0.999\right). \tag{15}$$

We compute $\bar{\alpha}_t$ from the clipped rates, giving terminal mask probability approximately 0.9999999903 without setting $\beta_T = 1$.

Denoising and sampling. Each example independently samples t_b from $1, \ldots, T$ uniformly. For the currently absorbed editable positions $\mathcal{A} = \{(b,i) : U_i^{(b)} = 1,\ z_{t_b,i}^{(b)} = m\}$,

$$\mathcal{L}_{\mathrm{abs}} = -\frac{1}{|\mathcal{A}|}\sum_{(b,i)\in\mathcal{A}} \log \pi_{\theta,i}\left(z_{0,i}^{(b)} \mid \mathbf{z}_{t_b}^{(b)}, U^{(b)}, t_b\right). \tag{16}$$

No other positions or variational-bound terms contribute. Empty-corruption minibatches are redrawn for up to four total attempts, then skipped if still empty. Sampling conditions on context

tokens from the zeroed-mask encoding, initializes editable tokens to m, and reveals each masked token at reverse step t with probability

$$\eta_t = \frac{\bar{\alpha}_{t-1} - \bar{\alpha}_t}{1 - \bar{\alpha}_t} = \frac{\bar{\alpha}_{t-1}\beta_t}{1 - \bar{\alpha}_t}. \tag{17}$$

Revealed codes are sampled from $\pi_{\theta,i}$ and remain fixed; context tokens are restored after each update. Reveal order is stochastic, not confidence-based. We set $\eta_1 = 1$ and terminate the 500-step schedule early if all editable tokens have been revealed, checking that none remain masked.

C.4 Training Settings and Computational Cost

Table 5 gives the two-stage settings. Loaders retain final partial batches. FP16 AMP uses dynamic loss scaling; nonfinite-gradient updates are skipped without advancing the optimizer schedule or EMA.

Early stopping and checkpoint selection. The absorbing, uniform, and codebook-similarity diffusion models share one early-stopping policy on the training objective, with a minimum of 50 epochs, patience of 30 epochs, and improvement threshold 10^{-6}. Each model monitors its own training objective, so training duration is set by this policy rather than a fixed number of epochs. Checkpoints are saved every 20 epochs. For CodePaint, training stops at epoch 82, and we use the epoch-80 checkpoint, after the training CE has plateaued, with EMA weights for inference.

Setting	VQ-VAE	Absorbing D3PM
Source tiles (train/test)	900 / 100	900 / 100
Layout–mask pairs	Not applicable	11,700 / 1,300
Optimizer	Adam	AdamW
Optimizer (β_1, β_2)	$(0.9, 0.999)$	$(0.9, 0.99)$
Learning rate	10^{-5}	Peak 2×10^{-4}
Learning-rate schedule	Constant	1,000-update warmup; 200-epoch cosine decay
Minimum learning rate	Not applicable	10^{-6}
Weight decay	0	10^{-4}
Batch size (train/test)	32 / 32	32 / 32
Precision	FP32	FP16 AMP
Gradient-norm clipping	None	1.0
EMA decay	None	0.9999
Training seed	42	42
Training input	Alternating 256×256 crops and 512×512 tiles	Cached 64×64 token grids
Codebook capacity / active codes	128 / 50	50 output classes
Parameters	4,109,249	50,745,522
Denoiser dropout	Not applicable	0.1
Diffusion length T	Not applicable	500
Noise schedule	Not applicable	Cosine, $s = 0.008$
Transition-rate clipping	Not applicable	$[10^{-8}, 0.999]$
Checkpoint-saving interval	20 epochs	20 epochs
Completed epochs	3,000 (fixed)	82 (early stopping)
Checkpoint used	Epoch 2,500	Epoch 80 (EMA for sampling)

Table 5: Two-stage ASAP7 configuration. Tokenizer epochs are cumulative; noise and learning-rate schedules refer to different processes.

Diffusion training uses one 24-GB NVIDIA RTX A5000 GPU and takes 1.42 hours for training, or approximately 1.45 hours including initialization and checkpoint saving.

C.5 Baseline and Ablation Training Settings

All inpainting models use the source split, fixed masks, and training seed described above. The transition variants share the frozen VQ-VAE, its 50-code vocabulary, denoiser backbone, and early-

stopping policy. Their AdamW hyperparameters, batch size, warmup, clipping, precision, and EMA follow Table 5; variant-specific objectives and corruption kernels are given below.

Pixel DDPM. The U-Net has base width 128 and multipliers $[1, 1, 2, 2, 4, 4]$, with two residual blocks per level. The diffusion process has 1,000 steps with linear $\beta_t \in [10^{-4}, 0.02]$, and the noise-prediction MSE weights missing and visible pixels equally. Training uses AdamW, learning rate 5×10^{-6}, zero weight decay, batch size 4, FP32, and gradient clipping 0.5. We use the epoch-19 checkpoint, after the training loss has plateaued.

KL-LDM. The autoencoder produces $64 \times 64 \times 64$ continuous latents. It trains on the same clean-source split for 3,000 epochs with Adam, learning rate 10^{-4}, batch size 32, and BCE plus 10^{-6}-weighted KL divergence. The epoch-2,472 checkpoint, taken after the reconstruction loss has plateaued, is frozen. The conditional U-Net uses concatenation and cross-attention, 400 epochs of AdamW, learning rate 10^{-4}, weight decay 0.01, and batch size 104. It uses Pixel DDPM's noise schedule with missing/visible-region MSE weights 4:1.

PatternPaint. The full SD2-inpaint U-Net is fine-tuned with an empty prompt and frozen VAE/text encoder. Training uses 15,000 AdamW updates, learning rate 10^{-5}, weight decay 0.01, effective batch size 4 (microbatch 1, accumulation 4), FP16, and gradient clipping 1.0. Noise-prediction MSE uses the pretrained 1,000-step scaled-linear schedule, $\beta_t \in [0.00085, 0.012]$. We use the update-12,500 checkpoint, after the training loss has plateaued.

Uniform and codebook-similarity D3PM. Both use $T = 500$ and corrupt only editable tokens during training, and their inference follows the zeroed-mask protocol above. For $K = 50$, the transitions are

$$\mathbf{Q}_t^{\text{uni}} = 0.5\mathbf{I}_K + \frac{0.5}{K}\mathbf{1}_K\mathbf{1}_K^{\mathsf{T}}, \qquad \mathbf{Q}_t^{\text{sim}} = 0.5\mathbf{I}_K + 0.5\mathbf{P}.$$

The similarity matrix $\mathbf{P}$ uses the frozen codebook's top-eight cosine neighbors, temperature 0.1, and Sinkhorn normalization. Both minimize editable-region clean-token CE, weighting changed tokens by two, with cosine learning-rate decay toward 10^{-6}. Similarity inference uses the epoch-35 EMA weights, after the training CE has plateaued.

FSQ tokenizer. The VQ-VAE encoder/decoder backbone is retained with a 64-to-2 projection, scalar levels $[8, 16]$ (128 possible tokens), straight-through rounding, and a 2-to-64 decoder projection. Pixel-BCE training uses Adam, batch size 32, and FP32: 10^{-4} for 944 epochs on full tiles, then 10^{-5} with alternating 256×256 crops and full tiles, for 3,000 epochs total. We use the epoch-2,960 checkpoint, after the training BCE has plateaued.

Categorical VAE / Gumbel–Softmax tokenizer. The backbone, data, batch size, precision, crop schedule, and BCE objective follow FSQ, with 128 categories and 64-dimensional embeddings. Training uses hard straight-through Gumbel–Softmax, with temperature decaying exponentially from 1.0 to 0.3 over 944 epochs, then fixed. Evaluation and encoding use deterministic argmax, and there is no KL or entropy regularizer. The retained recovery run uses Adam at 10^{-4} through epoch 95 and 10^{-5} thereafter. We use the epoch-2,300 checkpoint, after the training BCE has plateaued.

Diffusion with alternative tokenizers. Frozen FSQ/categorical tokenizers have 127/16 training-active codes. Their denoisers use the absorbing process, loss, zeroed-mask conditioning protocol, and optimization settings of CodePaint, including the 200-epoch learning-rate schedule and common early-stopping policy. The runs stop at epochs 120/115, and we use the epoch-80/40 checkpoints, respectively, after the training CE has plateaued, with EMA weights for inference.

C.6 Model Size and Sampling Cost

CodePaint has 54,854,771 parameters in total. Cohort generation uses one RTX A5000, batches of 32, the 500-step absorbing schedule, and FP16 denoising; FP32 VQ decoding uses sub-batches of four. Generating 1,300 samples takes 1,144 seconds (0.880 seconds/sample), including initialization, decoding, and writes, but excluding GDS conversion and DRC.

C.7 Sampling and Physical Verification

Cohort and decoding. Each of the 1,300 test pairs receives one stochastic completion, with no selection across generation rounds; all samples enter the clean-rate denominator before deduplication. For CodePaint, contexts are permuted with seed 1,234, and a batch whose first sample has zero-based position o uses seed $20{,}260{,}924 + o$. The selected EMA denoiser and frozen tokenizer produce binary metal rasters by decoding original codebook IDs and thresholding sigmoid occupancy probabilities at 0.5.

Evaluation branches. *Direct* uses decoded rasters without geometric cleanup; *+Denoise* applies PatternPaint's template-based cleanup before GDS conversion. Both branches use the same pixel-mask replacement.

GDS conversion and stitch-back. At $1\,\mathrm{nm}$ per pixel and $0.25\,\mathrm{nm}$ per DBU, each pixel spans four DBU per axis. Maximal horizontal metal runs are merged across consecutive rows into rectangles on GDS layer 19, datatype 0. Placement uses the source bounding box and reverses the image vertical axis. For source M1 geometry G, pixel region $\mathcal{M}$ expressed in source coordinates as Ω, and generated geometry $\widehat{G}$,

$$G_{\text{completed}} = (G \setminus \Omega) \cup (\widehat{G} \cap \Omega). \tag{18}$$

Geometry outside Ω is preserved even when editable token blocks extend beyond it. Hard replacement does not guarantee boundary continuity; resulting violations remain part of the DRC evaluation.

DRC scope and scoring. Calibre 2018.2_52.40 (Mentor Graphics, 2018) checks the complete stitched AES M1 design using ASAP7 r1p7 deck `drcRules_calibre_asap7_201222a.rul`. We evaluate nine single-layer checks on M1: `M1.W.1`, `M1.S.1`–`M1.S.6`, `M1.A.1`, and `GEOMETRY.NONORTHOGONAL`. The unchanged source design passes all nine. A completion is clean only when its run succeeds and reports zero violations across this set. Execution errors must be resolved before reporting the fixed-cohort rate. The per-rule limit of 1,000 results can truncate counts, but not zero-versus-nonzero classification. Raw summaries yield 984 clean and 316 violating CodePaint Direct samples, with no execution errors: $984/1{,}300 \approx 75.69\%$.

D RULE-TARGETED GENERATION DETAILS

This section gives the protocol behind Table 2 and breaks its pooled rates down by target.

D.1 CLIPS, TARGETS, AND REGENERATED NEIGHBORHOODS

Clip cohort. Every clip is a 512×512 nm ASAP7 M1 tile rasterized at 1 nm per pixel, taken from the test split of a set of tiles whose measured minimum tip-to-side spacing is exactly 25 nm. Each clip therefore contains one tip-to-side site at the rule minimum, which we call the site. For each target we scan this split in a fixed order and keep the first 150 clips whose site can be moved to the target. The six targets share most of their clips, and 110 clips are used at all six.

Editing the site to the target. For a target T we move the tip edge of the site along its axis until the tip-to-side gap equals T. The wire end is extended for $T < 25$ nm, left unchanged for $T = 25$ nm, and retracted for $T = 28$ nm, and nothing else in the clip changes. A clip is kept only if at least 4 pixels of the edited wire remain and the rule checker then finds the site at T within 1 nm. The edited clip is what the sampler receives, and it also serves as the reference for cleanup and for the metrics below.

Neighborhood B. The site box is the union of a strip along the tip, a strip along the facing side next to the tip, and the gap between them. B is this box enlarged $4\times$ about its center, widened to at least 64 nm per side, clipped to the tile, and rounded outward to the 8 nm token grid. It covers a median of 288 to 360 of the $64 \times 64 = 4{,}096$ tokens, depending on the target, because the gap grows with T. Every method regenerates exactly B and keeps all tokens outside it fixed.

Sampling. All CodePaint rows use the same pretrained model without retraining. Each clip receives four independent draws with a fixed sampling seed, giving $150 \times 4 = 600$ draws per method and target. Every draw is scored, with no selection among draws.

D.2 COMPARED METHODS

The three retention methods clamp exactly $k = 9$ tokens inside B to the tokens of the edited clip and sample the rest of B. *Random* draws its nine tokens uniformly from B, once per clip, so all four draws of a clip share the same mask. *BoxKeep* retains the 3×3 token block nearest the center of the site. *TokenProbe* retains the nine tokens with the highest score in Eq. 7, computed from perturbations of the tip by $\Delta = \{\pm1, \pm2, \pm3\}$ nm. The unconditional row regenerates all of B and retains nothing.

SCALE (Ho et al., 2026) conditions layout generation on text prompts, which in our setting specify the target tip-to-side spacing. We reimplement it with Qwen2.5-0.5B, trained by supervised fine-tuning for masked layout completion on ASAP7 tiles. We prompt it with a text description of the target rule and spacing, let it regenerate the same neighborhood B of the same clips, and draw four samples per clip. This baseline was run only with the $4\times$ neighborhood, which is why Tables 2 and 6 use that window.

D.3 CLEANUP, DESIGN-RULE CHECKING, AND METRICS

Cleanup. Every output of every method, including the SCALE row, goes through PatternPaint's template scan-line cleanup with a 2-pixel snapping threshold before GDS conversion. The template is the edited clip handed to the sampler. Because that template contains the site at the target spacing, cleanup can move a slightly displaced tip back onto the target line in every method alike.

Design-rule checking. Only B is replaced in the clip's source design, following Eq. 18. Calibre 2018.2_52.40 (Mentor Graphics, 2018) then runs the ASAP7 r1p7 deck of Appendix C.7 on the stitched design, and we count the violations of every rule in the deck. The reference for each clip is the edited clip encoded and decoded without sampling, passed through the same cleanup, stitching, and deck.

Metrics. A draw is *Kept* if its final raster contains at least one tip, an edge of at most 36 nm, facing a side, an edge longer than 36 nm, at exactly T along an unobstructed ray, with the gap region overlapping B with positive area. The pair need not be the original site, and Kept is measured on the cleaned raster without further filtering. A draw is + *clean* if it is Kept and no rule has more violations than on the clip's reference. A draw is + *new* if it is Kept and clean and its pixels inside B differ from the reference. All rates are over all 600 draws of a method and target, including draws that fail.

Diversity. H_1 (Eq. 3) is computed over each method's library of Kept and clean layouts pooled over the six targets, after removing layouts whose pixels inside B repeat another layout of the same clip. These libraries contain 4 layouts for SCALE, 84 for unconditional sampling, 133 for Random, 325 for BoxKeep, and 498 for TokenProbe. H_1 grows with library size, so its differences in Table 2 partly reflect how many valid layouts each method produces.

D.4 Results by Target

Table 6: Rule-targeted generation on ASAP7 `M1.S.2` across tip-to-side targets. Each CodePaint arm retains nine tokens of the $4\times$ neighborhood and they differ only in which nine: anywhere in the window (Random), the 3×3 block on the site (BoxKeep), or the nine highest-scoring tokens of Eq. 7 (TokenProbe). The unconditional row retains nothing. The SCALE row is our reimplementation of Ho et al. (2026), which uses text prompts to condition the model on the target spacing. Kept requires the exact target gap in the regenerated window. Kept + clean additionally requires no DRC violation beyond the clip's source reference. Kept + clean + new further requires the completion to differ from that reference. Best value in each column is bold.

Method	Target tip-to-side gap (nm)						Mean
	15	18	20	22	25	28	
Kept (%) ↑							
SCALE (Ho et al., 2026)	2.5	33.8	4.8	5.8	8.7	2.8	9.7
CodePaint, unconditional	0.7	11.0	5.3	6.0	24.0	12.8	10.0
CodePaint + Random	1.5	12.5	6.3	8.3	32.3	17.7	13.1
CodePaint + BoxKeep	27.3	33.0	30.3	38.5	52.5	51.5	38.9
CodePaint + TokenProbe	**40.7**	**58.8**	**50.3**	**61.0**	**56.3**	**64.3**	**55.2**
Kept + clean (%) ↑							
SCALE (Ho et al., 2026)	0.0	0.3	0.0	0.0	0.2	0.2	0.1
CodePaint, unconditional	0.0	2.5	1.5	2.0	10.7	5.5	3.7
CodePaint + Random	0.7	2.8	2.8	3.0	16.0	9.0	5.7
CodePaint + BoxKeep	9.5	9.3	12.7	17.0	19.2	20.5	14.7
CodePaint + TokenProbe	**16.8**	**22.7**	**19.2**	**25.2**	**25.0**	**26.2**	**22.5**
Kept + clean + new (%) ↑							
SCALE (Ho et al., 2026)	0.0	0.3	0.0	0.0	0.2	0.2	0.1
CodePaint, unconditional	0.0	2.5	1.3	1.3	5.7	2.2	2.2
CodePaint + Random	0.7	2.7	2.8	2.0	7.2	4.3	3.3
CodePaint + BoxKeep	5.7	5.7	9.3	9.0	10.3	12.8	8.8
CodePaint + TokenProbe	**9.3**	**11.5**	**12.3**	**12.8**	**14.5**	**14.2**	**12.4**

Table 6 reports each target separately, and its Mean column reproduces Table 2. TokenProbe gives the highest rate at every target for all three metrics. Its Kept + clean rate ranges from 16.8% at 15 nm to 26.2% at 28 nm, against 9.3% to 20.5% for BoxKeep. Unconditional sampling reaches a noticeable Kept + clean rate only at the legal targets 25 and 28 nm (10.7% and 5.5%) and stays at or below 2.5% for the violating targets 15 to 22 nm. Random retention adds at most 5.3 points over unconditional sampling at any target, whereas BoxKeep and TokenProbe raise every violating target well above it. The SCALE row realizes the 18 nm target in 33.8% of draws but almost never without new violations, reaching at most 0.3% Kept + clean at any target.

E FEW-SHOT CROSS-PDK ADAPTATION: EXPERIMENTAL SETTING

This appendix details the ASAP7-to-commercial-16 nm adaptation study in Table 3.

E.1 TARGET-PDK DATA

An AES block synthesized and placed-and-routed in the target technology provides approximately $100 \times 100\,\mu$m of routed M1 geometry. Clips are 512×512 binary rasters at 1 nm per pixel, matching both the manufacturing grid and GDS database unit. The routed block and the identity stitch-back control are DRC-clean under the evaluation regime below.

We adapt on nested subsets of $N \in \{20, 50, 100\}$ target-PDK clips and evaluate on 100 contexts from a disjoint region. Each clip uses the 13 fixed masks in Appendix B.4, yielding $13N$ training records and 1,300 evaluation pairs.

E.2 CODEPAINT ADAPTATION

The encoder and codebook remain frozen. The decoder and denoiser are initialized from the ASAP7 checkpoints (Appendix C.4), and Table 7 lists their training settings.

Decoder adaptation. Each update uses 8 random 448×448 target-clip crops with random pixel offsets and mirror orientations. The decoder minimizes binary cross-entropy against these crops using tokens from the frozen encoder. A source-preservation term, weighted by $1/3$, uses 8 ASAP7 token grids with the frozen decoder's binarized outputs as reconstruction targets.

Denoiser adaptation. Each target clip yields 32 variants: eight pixel shifts (the identity and seven distinct offsets in $\{0, \ldots, 7\}^2$), each with four mirror orientations. Exposed borders use edge replication. Re-encoding these variants and applying the 13 masks produces $416N$ instance records. For the rehearsal corpus $\mathcal{D}_{\text{tgt}}^{\text{bal}}$, the $13N$ unaugmented records are grouped by scan-line topology (C_x, C_y), retaining at most four records per topology (76, 192, and 364 grids for $N = 20$, 50, and 100).

Training follows Eq. 8 with $\lambda = 1$ and separate, independently sampled instance and rehearsal minibatches under the target-PDK condition.

Setting	Decoder adaptation	Denoiser adaptation
Initialization	ASAP7 tokenizer	ASAP7 denoiser
Trainable parameters	2.1 M	50.8 M
Target records	N clips, random crops	$416N$
Auxiliary term	ASAP7 grids, weight $1/3$	$\mathcal{D}_{\text{tgt}}^{\text{bal}}$, $\lambda = 1$
Optimizer	Adam	AdamW $(0.9, 0.99)$
Learning rate	2×10^{-4}	2×10^{-5} (2×10^{-4} for PDK parameters)
Schedule	Cosine	200-update warmup, cosine to 10%
Weight decay	0	0.01
Batch	8 + 8	16 + 16
Updates	8,000	48,000 (all N)
Gradient clipping	None	1.0
Precision	FP32	FP16 AMP
EMA decay	None	0.999 (used for sampling)
Seed	1234	1234
GPU, wall clock	TITAN RTX, <15 min	TITAN RTX, 2.45 h

Table 7: CodePaint training settings for the few-shot cross-PDK study. Batch sizes list target and auxiliary samples per update.

E.3 EVALUATION

Baseline. We compare CodePaint against PatternPaint (Zhou et al., 2025b), an SD2-inpaint model that denoises in a continuous VAE latent space. It reuses the model and optimizer settings of Appendix C.5, fine-tuned from the public SD2-inpaint checkpoint on the same $13N$ target records. Its zero-shot row applies this checkpoint without any layout training. Its decoded output is binarized at 0.5, and the known region is restored from the input at pixel resolution before scoring, as for CodePaint.

Sampling and DRC. Each method generates one sample per context–mask pair, giving 1,300 samples per setting, and CodePaint samples at temperature 1. Each generated fill replaces its masked region and is reinserted into the routed block with a 1 μm halo of real geometry. Calibre (Mentor Graphics, 2018) then checks 99 M1 rule entries from the foundry's 16 nm deck, fixed before any model was evaluated and including double-patterning decomposability. Windows are separated by a 500 nm guard band so that each violation can be attributed to a single window. A sample is DRC-clean when its window passes every evaluated check. Direct reports the clean rate of raw samples, and +Denoise reports it after the template cleanup in Appendix C.7.

Diversity. Following Zhou et al. (2025b), H_1 is the entropy of the scan-line topology (C_x, C_y), measured inside the masked region, over the distinct DRC-clean fills after cleanup. Fills are deduplicated by their squish representations. The resulting libraries contain 254, 337, and 403 patterns for CodePaint and 64, 78, and 130 for PatternPaint at $N = 20$, 50, and 100, respectively.

F Additional Results

We present additional qualitative results on DRC-clean layout completion, sampling diversity, and pixel noise. The comparisons include Pixel DDPM, KL-LDM, PatternPaint (fine-tuned SD2), and Absorbing D3PM (ours). *Original layout* denotes the complete source layout before masking. Within each row, all outputs share the same source tile and missing-region mask. Blue shading in the input marks the missing region; dotted blue outlines identify that region in the remaining columns.

All displayed completions contain generated geometry within the missing region and are shown without geometric cleanup. Sampling and conditioning follow the main experiments, as described in the implementation appendix. The examples are selected to illustrate specific outcomes and complement the aggregate evaluation; they do not estimate overall success rates.

F.1 Model Size and Inference Efficiency

Table 8: Model size and inference efficiency on the same NVIDIA RTX A5000. All methods use exactly 250 denoiser evaluations (NFE) with FP16 denoisers; auxiliary modules and stored-weight precision remain model-specific. Batch size is 32, except Pixel DDPM, which uses 16 after out-of-memory at 32. Results average seven synchronized measurements after two warmup runs. Time per image is mean batch latency divided by batch size. Speedup is throughput relative to SD2-inpaint. Memory is peak PyTorch-allocated GPU memory. Loading, file writes, cleanup, GDS conversion, and DRC are excluded.

Method	Setup		Inference			
	Params. (M) ↓	Batch	Time (s/img) ↓	Throughput (img/s) ↑	Speedup ↑	Peak mem. (GiB) ↓
PatternPaint (SD2-inpaint)	1289.97	32	3.773	0.265	1.00×	18.64
Pixel DDPM	93.37	16	15.056	0.066	0.25×	13.52
KL-LDM	72.11	32	0.618	1.617	6.10×	1.24
Uniform	54.85	32	0.485	2.062	7.78×	1.36
CodePaint (ours)	54.85	32	0.529	1.891	7.13×	1.34

Table 8 compares model size and inference efficiency on an NVIDIA RTX A5000 using 250 denoiser evaluations and FP16 denoisers. Batch size is 32 for all methods except Pixel DDPM, which uses 16 due to GPU memory limits; auxiliary modules and stored-weight precision remain model-specific. The reported time per image measures amortized batch latency, rather than single-image latency.

Compared with PatternPaint, CodePaint reduces the parameter count from $1{,}289.97$M to 54.85M and achieves $7.13\times$ higher throughput, while reducing peak allocated GPU memory from 18.64 to 1.34 GiB. Relative to Uniform, CodePaint has the same parameter count and similar peak memory usage, with slightly lower throughput (1.891 versus 2.062 images/s). These measurements cover model inference only and exclude geometric post-processing and DRC.

F.2 DRC-clean layout completion

Figure 6 compares completions on five different source tiles. Our displayed outputs pass DRC, while each displayed baseline output contains design-rule violations. Verification replaces only the masked M1 region in the complete source design and evaluates all single-layer M1 checks, as in the main evaluation. Here, *DRC clean* means zero violations across all nine checks (Appendix C.7). At most five representative Calibre (Mentor Graphics, 2018) violation boxes are shown per output, with corresponding rule labels below the layout.

F.3 Sampling diversity

Figure 7 shows two stochastic completions from our model for each of four inputs. Each pair uses an identical input and mask but produces different geometry inside the missing region, while preserving the surrounding layout. Both outputs in every pair pass all single-layer M1 checks. These examples illustrate that the model can generate multiple DRC-clean alternatives for the same completion task.

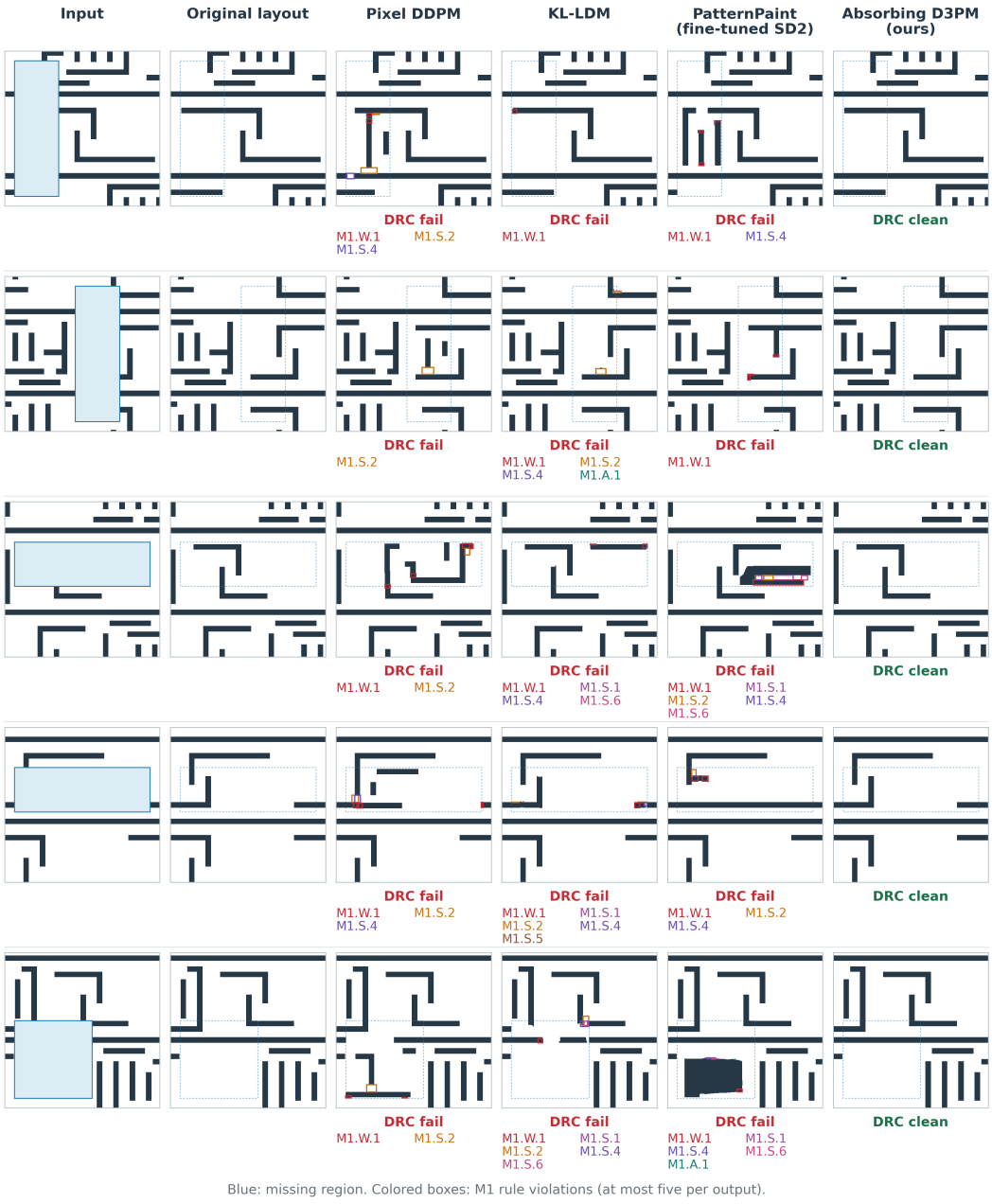


Figure 6: DRC-clean completion on five inputs. Each row shows the input, original layout, and four model outputs. Our completions pass all single-layer M1 checks. Colored boxes identify up to five violations per baseline output, with rule labels matched by color.

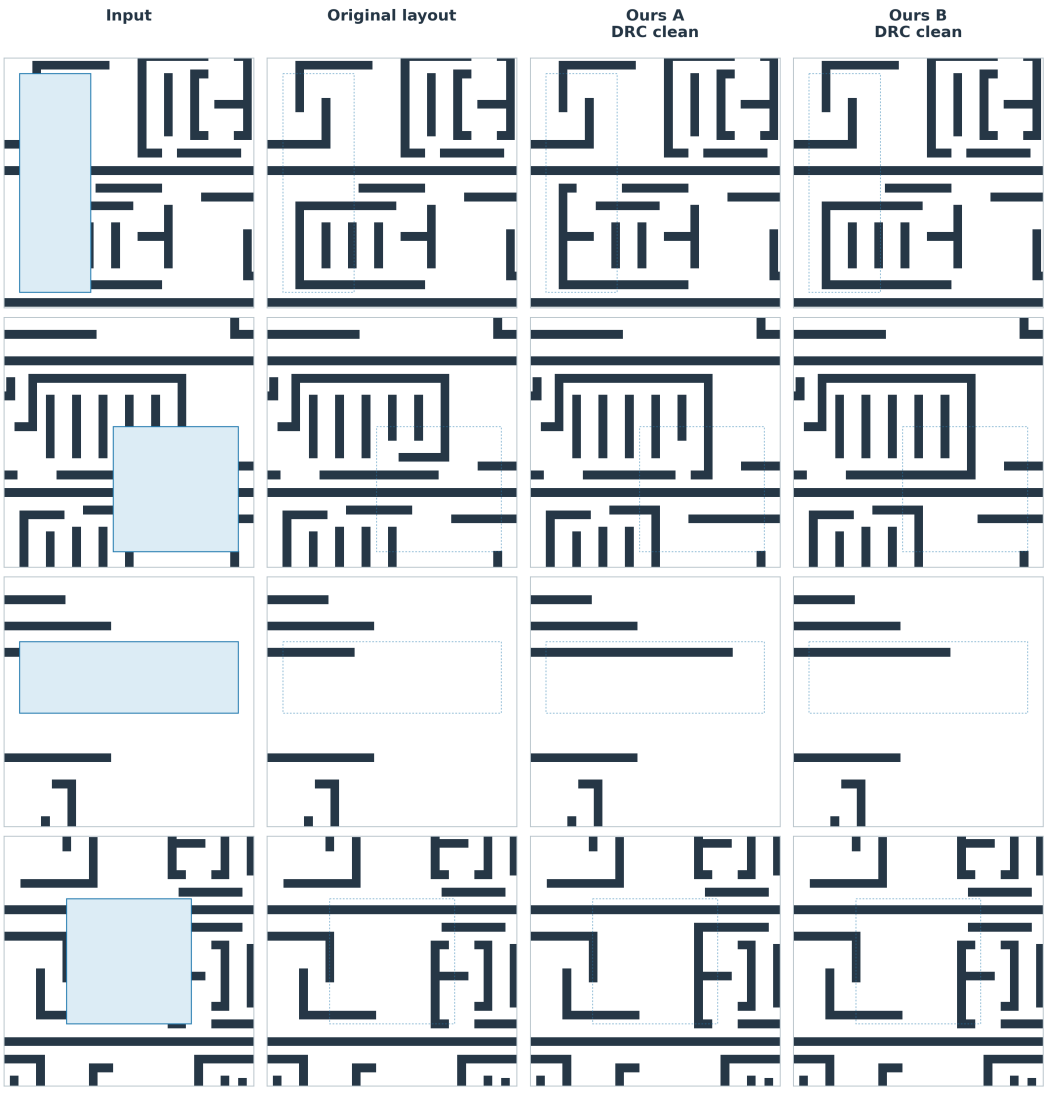


Figure 7: Diverse DRC-clean completions from Absorbing D3PM. Each row shows one input, its original layout, and two distinct outputs sampled using the same input and mask.

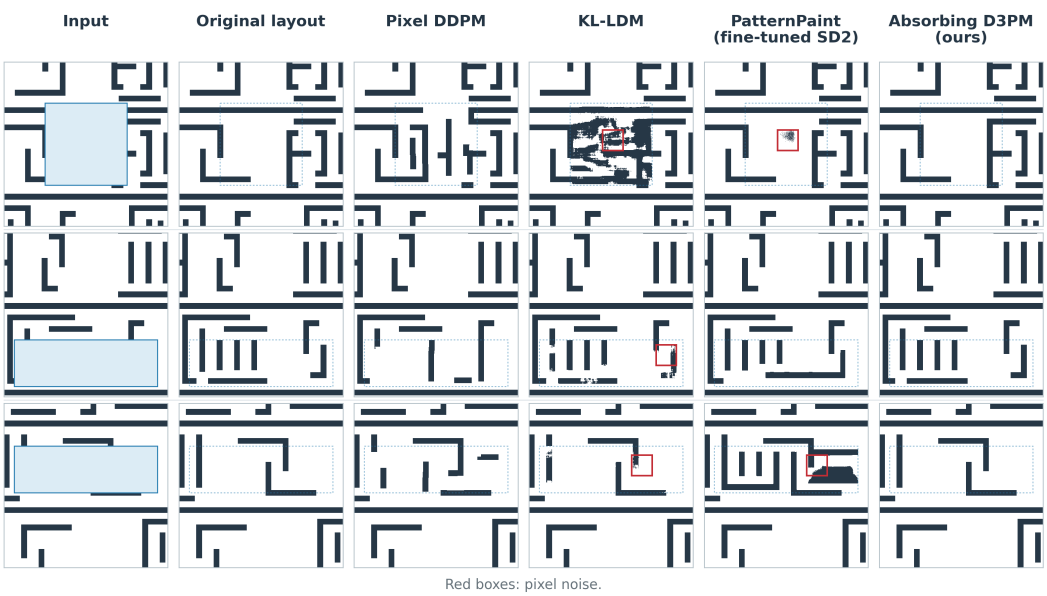


Figure 8: Pixel-noise comparison on three inputs. Red boxes mark representative noise regions in KL-LDM and PatternPaint, with at most one box per output. Our completions show no isolated-pixel defects.

F.4 Pixel noise

Figure 8 compares three additional inputs with visible pixel artifacts in the baseline outputs. Speckling and irregular boundaries are most apparent in KL-LDM and PatternPaint, whereas our displayed completions contain no isolated-pixel defects and pass all single-layer M1 checks. Pixel DDPM also has little isolated-pixel noise in these examples. Red boxes highlight at most one noise region per baseline output directly on the layout; these boxes identify visible artifacts rather than Calibre (Mentor Graphics, 2018) violation bounds.