Efficient and Training-Free Single-Image Diffusion Models

Supplementary Material

Contents

S1. Connections to Prior Single-Image Generative Methods

In this section, we discuss the connections and differences between our method and prior single-image generative approaches. Our method is a multi-scale framework that iteratively refines image patches at each scale, drawing inspiration from SinGAN [58], GPNN [26], GPDM [21], and SinDDM [38]. As in these works, patches at coarser scales govern the global structure of the generated image while patches at finer scales control its texture. Additionally, iterative patch refinement is performed at each scale, similar to previous work. However, our approach also differs from each of these in key ways, as we describe in the following sections.

S1.1. Comparison with GPNN

Both GPNN [26] and our method employ coarse-to-fine sampling, iteratively refining patches and reassembling them into an image. GPNN only injects noise once at the coarsest-scale initialization and then performs hard patch replacement with the (single) nearest neighbor at each iteration. In contrast, we use diffusion as the patch-selection mechanism, which employs an optimal diffusion denoiser that operates via a weighted average of all patches, and also injects noise at every sampling step. As the optimal denoiser approaches the final step (i.e., $\sigma(t) \to 0$), we also select or copy a single clean patch; however, all previous intermediate iterations of soft patch selection ground the probabilistic sampling for the patch prior p_{data}. The one-time mild noise injection in GPNN causes samples to frequently duplicate much of the reference image—behavior that we do not observe in our approach. We provide an analysis of this effect in Sec. S5, Figure S7, and Table S1.

S1.2. Comparison with GPDM

GPDM [21] has a similar multi-scale formulation as GPNN, but it uses a very different patch generation mechanism based on an iterative sliced Wasserstein distance (SWD) optimization. Specifically, GPDM aims to optimize the Wasserstein distance

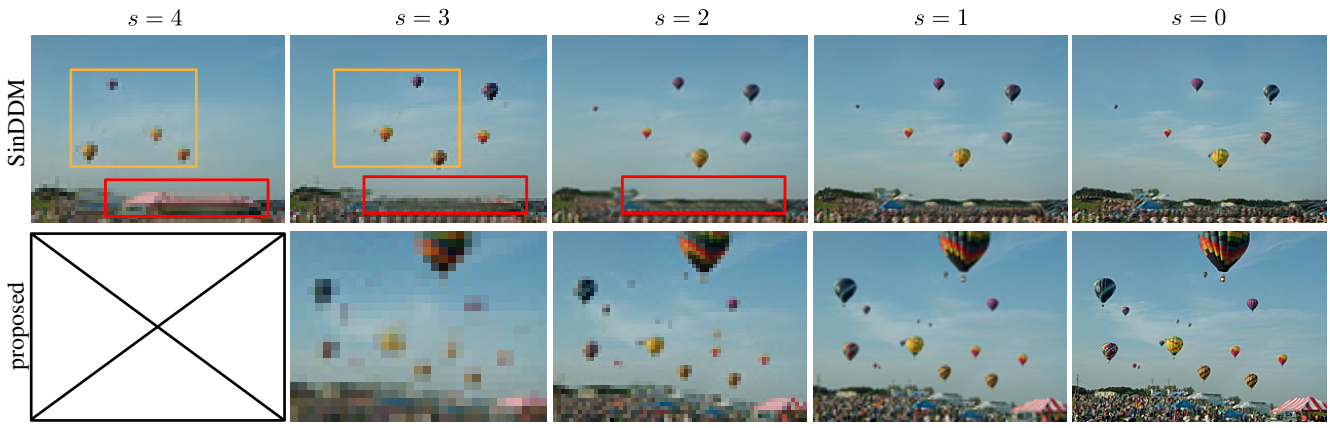


Figure S1. Multi-scale consistency comparison vs. SinDDM. SinDDM outputs (top) show significant inconsistency across scales, while our method (bottom) produces consistent outputs across scales.

between the generated patch distribution and the reference patch distribution, where SWD provides a tractable approximation to the Wasserstein distance. Our diffusion formulation shares the same goal, namely to sample a patch distribution that matches that of the reference. However, diffusion naturally enables the incorporation of various guidance and sampling techniques, which may not be straightforward to implement within the SWD optimization framework.

S1.3. Comparison with SinDDM

SinDDM [38] is a multi-scale trained diffusion model that performs patch-level generation by restricting the receptive field of a convolutional neural network (CNN). The most notable difference in our method is that it does not require training, but instead uses the closed-form denoiser. In addition, the exact multi-scale formulation also differs in the following ways. (1) We use a forward process with no explicit blurring (only noising), and (2) we use Laplacian blending across scales instead of scheduled linear interpolation to better preserve coarse-scale structure (e.g., compare Algorithm 2, L9,16–17 to SinDDM Sec. 3.1, Algorithm 2, L8). Qualitatively, our outputs across coarse scales are more consistent than those of SinDDM (Figure S1). Consistency across scales is a very important property for applications such as structural analogies, where the structure image is only injected at the coarsest scale.

S2. Closed-Form Diffusion Denoiser

S2.1. Derivation of the Closed-Form Diffusion Denoiser

It is known that the diffusion denoiser or the score estimator can be computed in closed form when having access to the empirical distribution, i.e., the real dataset [36, 56]. Here, we adopt a similar derivation approach as outlined in Appendix B.3 of [36], with adjustments tailored to our specific objective function and the formulation of diffusion models.

Let us recall that the data distribution is defined as $p_{\text{data}}(\mathbf{y}) = \frac{1}{Y}\sum_{j=1}^{Y}\delta\left(\mathbf{y}-\mathbf{y}^{(j)}\right)$ for a dataset $\mathcal{Y} = \left\{\mathbf{y}^{(1)},\dots,\mathbf{y}^{(Y)}\right\} \subseteq \mathbb{R}^d$, and the forward perturbation process of $\mathbf{x}_t = \alpha_t\mathbf{y} + \sigma_t\boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ when given a specific clean signal $\mathbf{y}$, *i.e.*, $p(\mathbf{x}_t \,|\, \mathbf{y}) = \mathcal{N}\left(\mathbf{x}_t; \alpha_t\mathbf{y}, \sigma_t^2\mathbf{I}\right)$. We can derive that the marginal distribution $p_t(\mathbf{x}_t)$ of perturbed data at some $t \in [0,T]$ is a mixture of Gaussians.

$$
\begin{aligned}
p_t(\mathbf{x}_t) &= \int_{\mathbb{R}^d} p(\mathbf{x}_t,\mathbf{y})\,\mathrm{d}\mathbf{y} && \text{(S1)}\\
&= \int_{\mathbb{R}^d} p(\mathbf{x}_t \,|\, \mathbf{y})\, p_{\text{data}}(\mathbf{y})\,\mathrm{d}\mathbf{y} && \text{(S2)}\\
&= \int_{\mathbb{R}^d} p(\mathbf{x}_t \,|\, \mathbf{y}) \left(\frac{1}{Y}\sum_{j=1}^{Y}\delta(\mathbf{y}-\mathbf{y}^{(i)})\right)\mathrm{d}\mathbf{y} && \text{(S3)}\\
&= \frac{1}{Y}\sum_{j=1}^{Y}\int_{\mathbb{R}^d} p(\mathbf{x}_t \,|\, \mathbf{y})\delta(\mathbf{x}-\mathbf{y}^{(i)})\,\mathrm{d}\mathbf{y} && \text{(S4)}\\
&= \frac{1}{Y}\sum_{j=1}^{Y} p(\mathbf{x}_t \,|\, \mathbf{y}=\mathbf{y}^{(j)}) && \text{(S5)}\\
&= \frac{1}{Y}\sum_{j=1}^{Y}\mathcal{N}(\mathbf{x}_t;\alpha_t\mathbf{y}^{(i)},\sigma_t^2\mathbf{I}), && \text{(S6)}
\end{aligned}
$$

Then, recall that the training objective at a specific t is

$$
\begin{aligned}
\mathcal{L}(D;t) &= \mathbb{E}_{\mathbf{y}\sim\mathcal{Y},\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[w_t\|D(\alpha_t\mathbf{y}+\sigma_t\boldsymbol{\epsilon},t)-\mathbf{y}\|_2^2\right] && \text{(S7)}\\
&= \frac{1}{Y}\sum_{j=1}^{Y}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[w_t\|D(\alpha_t\mathbf{y}^{(j)}+\sigma_t\boldsymbol{\epsilon},t)-\mathbf{y}^{(j)}\|_2^2\right] && \text{(S8)}\\
&= \frac{1}{Y}\sum_{j=1}^{Y}\mathbb{E}_{\mathbf{x}_t\sim\mathcal{N}(\mathbf{x}_t;\alpha_t\mathbf{y}^{(j)},\sigma_t^2\mathbf{I})}\left[w_t\|D(\mathbf{x}_t,t)-\mathbf{y}^{(j)}\|_2^2\right] && \text{(S9)}\\
&= \frac{1}{Y}\sum_{j=1}^{Y}\int_{\mathbb{R}^d}\mathcal{N}(\mathbf{x}_t;\alpha_t\mathbf{y}^{(j)},\sigma_t^2\mathbf{I})w_t\|D(\mathbf{x}_t,t)-\mathbf{y}^{(j)}\|_2^2\,\mathrm{d}\mathbf{x}_t && \text{(S10)}\\
&= \int_{\mathbb{R}^d}\frac{w_t}{Y}\sum_{j=1}^{Y}\mathcal{N}(\mathbf{x}_t;\alpha_t\mathbf{y}^{(j)},\sigma_t^2\mathbf{I})\|D(\mathbf{x}_t,t)-\mathbf{y}^{(j)}\|_2^2\,\mathrm{d}\mathbf{x}_t, && \text{(S11)}
\end{aligned}
$$

which minimizes the loss for all possible input $\mathbf{x}_t$ for the denoiser. When given a specific input $\mathbf{x}_t$, the loss is

$$
\mathcal{L}(D;\mathbf{x}_t,t) = \frac{w_t}{Y}\sum_{j=1}^{Y}\mathcal{N}(\mathbf{x}_t;\alpha_t\mathbf{y}^{(j)},\sigma_t^2\mathbf{I})\|D(\mathbf{x}_t,t)-\mathbf{y}^{(j)}\|_2^2. \tag{S12}
$$

To find the denoiser that produces the minimum mean squared error estimate, we set the gradient of the loss with respect to

the denoiser to be $\mathbf{0}$:

$$\mathbf{0} = \nabla_D \mathcal{L}(D; \mathbf{x}_t, t) \tag{S13}$$

$$= \nabla_D \frac{w_t}{Y} \sum_{j=1}^{Y} \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{y}^{(j)}, \sigma_t^2 \mathbf{I}) \|D(\mathbf{x}_t, t) - \mathbf{y}^{(j)}\|_2^2 \tag{S14}$$

$$= \frac{w_t}{Y} \sum_{j=1}^{Y} \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{y}^{(j)}, \sigma_t^2 \mathbf{I}) \nabla_D \|D(\mathbf{x}_t, t) - \mathbf{y}^{(j)}\|_2^2 \tag{S15}$$

$$= \frac{2w_t}{Y} \sum_{j=1}^{Y} \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{y}^{(j)}, \sigma_t^2 \mathbf{I}) \left(D(\mathbf{x}_t, t) - \mathbf{y}^{(j)} \right) \tag{S16}$$

$$= \frac{2w_t}{Y} \left(\sum_{j=1}^{Y} \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{y}^{(j)}, \sigma_t^2 \mathbf{I}) \; D(\mathbf{x}_t, t) \; - \; \sum_{j=1}^{Y} \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{y}^{(j)}, \sigma_t^2 \mathbf{I}) \; \mathbf{y}^{(j)} \right). \tag{S17}$$

Then, we rearrange the equation to obtain that the denoiser that minimizes the loss at the noise level t, given a specific noisy input $\mathbf{x}_t$, is

$$D(\mathbf{x}_t, t) \; = \; \frac{\sum_{j=1}^{Y} \mathcal{N}\left(\mathbf{x}_t; \alpha_t \mathbf{y}^{(j)}, \sigma_t^2 \mathbf{I}\right) \; \mathbf{y}^{(j)}}{\sum_{j=1}^{Y} \mathcal{N}\left(\mathbf{x}_t; \alpha_t \mathbf{y}^{(j)}, \sigma_t^2 \mathbf{I}\right)} \; = \; \frac{\sum_{j=1}^{Y} \exp\left(-\frac{1}{2\sigma_t^2} \left\|\mathbf{x}_t - \alpha_t \mathbf{y}^{(j)}\right\|_2^2\right) \; \mathbf{y}^{(j)}}{\sum_{j=1}^{Y} \exp\left(-\frac{1}{2\sigma_t^2} \left\|\mathbf{x}_t - \alpha_t \mathbf{y}^{(j)}\right\|_2^2\right)}. \tag{S18}$$

Several other interpretations of this formula include kernel regression:

$$D(\mathbf{x}_t, t) = \frac{\sum_{j=1}^{Y} k_{\sigma_t}\left(\mathbf{x}_t, \alpha_t \mathbf{y}^{(j)}\right) \; \mathbf{y}^{(j)}}{\sum_{j=1}^{Y} k_{\sigma_t}\left(\mathbf{x}_t, \alpha_t \mathbf{y}^{(j)}\right)}, \tag{S19}$$

where $k_\sigma(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|_2^2}{2\sigma^2}\right)$ denotes the Gaussian kernel with bandwidth $2\sigma^2$. Another equivalent interpretation is as the empirical posterior mean based on the finite data samples:

$$D(\mathbf{x}_t, t) = \frac{\sum_{j=1}^{Y} p(\mathbf{x}_t \,|\, \mathbf{y} = \mathbf{y}^{(j)}) \, \mathbf{y}^{(j)}}{\sum_{j=1}^{Y} p(\mathbf{x}_t \,|\, \mathbf{y} = \mathbf{y}^{(j)})} \tag{S20}$$

$$= \sum_{j=1}^{Y} \frac{p(\mathbf{x}_t \,|\, \mathbf{y} = \mathbf{y}^{(j)})}{\sum_{j'=1}^{Y} p(\mathbf{x}_t \,|\, \mathbf{y} = \mathbf{y}^{(j')})} \mathbf{y}^{(j)} \tag{S21}$$

$$= \sum_{j=1}^{Y} \frac{p(\mathbf{x}_t \,|\, \mathbf{y} = \mathbf{y}^{(j)}) \, p(\mathbf{y} = \mathbf{y}^{(j)})}{\sum_{j'=1}^{Y} p(\mathbf{x}_t \,|\, \mathbf{y} = \mathbf{y}^{(j')}) \, p(\mathbf{y} = \mathbf{y}^{(j')})} \mathbf{y}^{(j)} \tag{S22}$$

$$= \sum_{j=1}^{Y} \frac{p(\mathbf{x}_t, \mathbf{y} = \mathbf{y}^{(j)})}{p(\mathbf{x}_t)} \mathbf{y}^{(j)} \tag{S23}$$

$$= \mathbb{E}_{\mathbf{y} \sim p(\mathbf{y} \,|\, \mathbf{x}_t)}[\mathbf{y} \,|\, \mathbf{x}_t] \,. \tag{S24}$$

Finally, we can introduce a softmax notation:

$$D(\mathbf{x}_t, t) = \sum_{j=1}^{Y} \frac{\exp\left(-\frac{1}{2\sigma_t^2} \left\|\mathbf{x}_t - \alpha_t \mathbf{y}^{(j)}\right\|_2^2\right)}{\sum_{j'=1}^{Y} \exp\left(-\frac{1}{2\sigma_t^2} \left\|\mathbf{x}_t - \alpha_t \mathbf{y}^{(j')}\right\|_2^2\right)} \mathbf{y}^{(j)} = \sum_{j=1}^{Y} \operatorname{softmax}\left(\left[-\frac{1}{2\sigma_t^2} \left\|\mathbf{x}_t - \alpha_t \mathbf{y}^{(j')}\right\|_2^2 \right]_{j'} \right)_j \mathbf{y}^{(j)} \tag{S25}$$

where $\left[-\frac{1}{2\sigma_t^2} \left\|\mathbf{x}_t - \alpha_t \mathbf{y}^{(j')}\right\|_2^2\right]_{j'}$ denotes a Y-dimensional vector. We will see in Sec. S3.2 that this helps connect the closed-form denoiser to the attention operation.

S2.2. Computability and Generalization of Closed-Form Denoisers

Closed-form denoisers [5, 8, 35, 36, 41, 49, 56, 64] have been extensively analyzed, but so far have seen limited practical use. Two challenges are commonly raised: *computability* and generalization/memorization. Our setting provides a sweet spot where both concerns are alleviated.

First, the full closed-form denoiser is computationally infeasible when applied to an entire internet-level image dataset, where each data point is itself a very high-dimensional vector and the summation involves millions of terms. In contrast, our method operates at the *patch level* of a *single* image, where each patch is small and the dataset consists of a finite number of patches. With the acceleration techniques introduced in this paper, closed-form denoising becomes practical and efficient.

Second, closed-form denoisers encode the prior as a sum of Dirac delta densities, so when $\sigma_t \to 0$, the distribution collapses onto a single term, which can raise concerns about memorization or overfitting when applied at the full-image level. In our case, however, the denoiser operates on *patches* rather than entire images, and our image sampler stitches these patchwise predictions into a rich and diverse distribution over global images. This is supported by both our diversity metrics (Table 1) and the unconditional generations shown in Figure S8.

Our observations connect naturally to the findings of Niedoba et al. [49] and Kamb et al. [35], which argue that neural network-based diffusion models generalize by performing locally optimal denoising operations across the training distribution. In contrast to their dataset-level analysis, we study this behavior in the *single-image* regime, in the spirit of the SinGAN and SinDDM literature, and show that patch-level closed-form denoising can be used to support a wide range of practical single-image applications, including unconditional generation, text-guided stylization, structural analogy, symmetrization and retargeting.

S3. Supplementary Implementation Details

In this section, we provide additional implementation details and describe several complementary techniques used to accelerate the patch-based closed-form denoiser. Specifically, Sec. S3.1 outlines how we batch-denoise patches using straightforward PyTorch operations; Sec. S3.2 shows how closed-form denoising maps onto the attention operation, enabling the use of fused kernels such as FlashAttention [13, 14, 57] for substantial speedups; Sec. S3.3 explains how FLUX's variational autoencoder (VAE) [39] spatially compresses RGB images into latent representations, providing a large constant-factor acceleration; Sec. S3.4 describes how approximate nearest-neighbour search further improves asymptotic runtime; and Sec. S3.5 details how all of these techniques combine to enable efficient generation of gigapixel-resolution images.

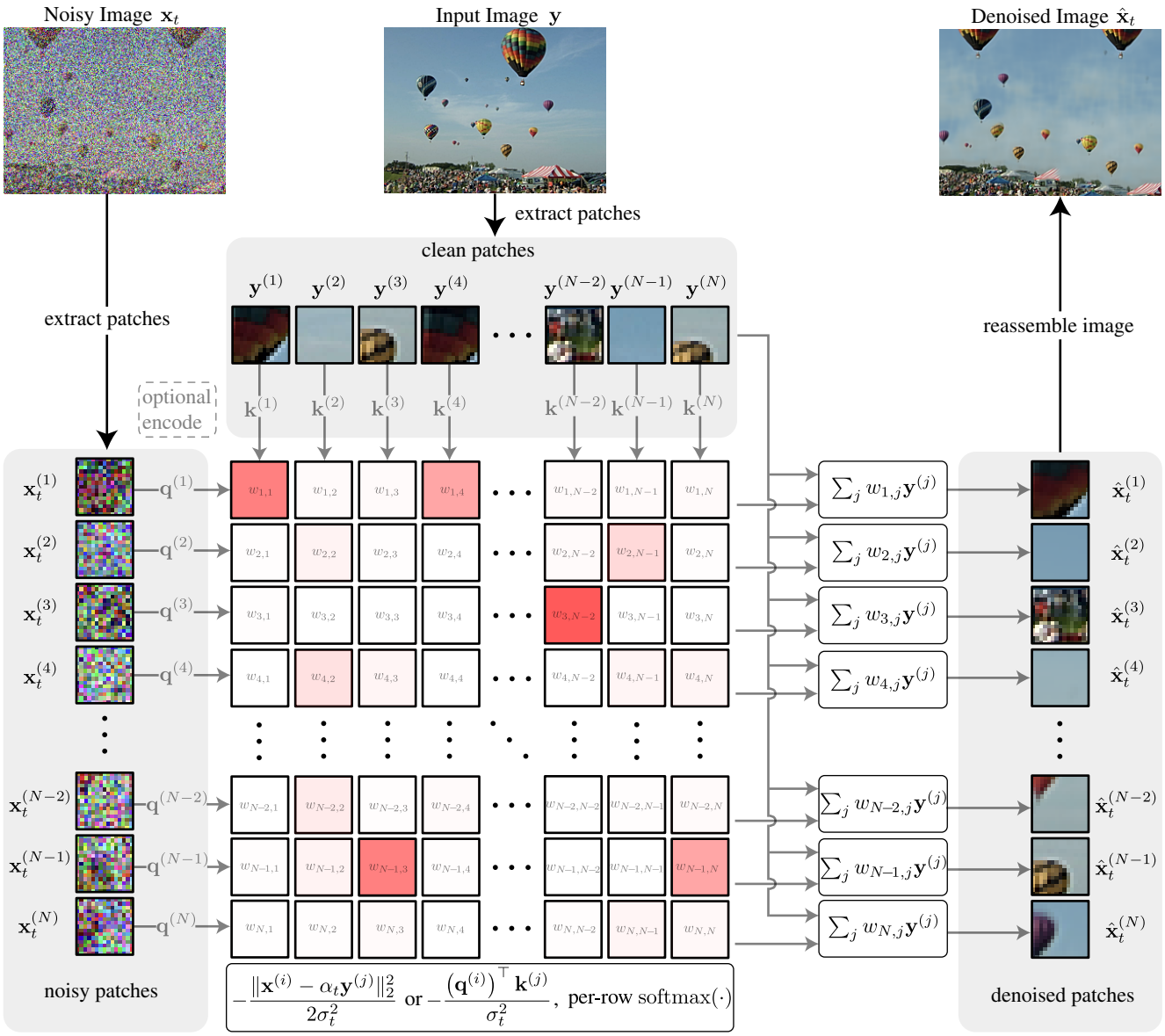


Figure S2. Overview of the batched closed-form denoising implementation. This figure illustrates both the naive vectorized PyTorch implementation in Sec. S3.1 and the version in Sec. S3.2 that implements this operation using attention. We assume the number of clean and noisy patches is the same ($Y = N$). To denoise a noisy image from $\hat{\mathbf{x}}_t$ to $\hat{\mathbf{x}}$, we extract both the noisy patches $\mathbf{x}_t^{(i)}$ and clean patches $\mathbf{y}^{(j)}$ and compute a weight matrix whose (i, j)-th entry $w_{i,j}$ predicts each denoised patch as a weighted sum of the clean patches. Finally, we assemble the denoised patches back to their original positions to obtain the denoised image.

S3.1. Naive Closed Form Denoiser Implementation

Vectorization. Sec. S2 gives an overview of closed-form denoising of a single data sample. In a single diffusion sampling step, given a noisy image $\mathbf{x}_t$, we extract a batch of noisy patches $\mathbf{x}_t^{(i)}$ to be denoised. Because these patches are relatively low dimensional, we can denoise them in parallel and fully leverage GPU acceleration. An illustration of the batched computation appears in Figure S2. Specifically, we stack the denoised patches $\hat{\mathbf{x}}_t^{(i)} = D(\mathbf{x}^{(i)}, t)$, the noisy patches $\mathbf{x}_t^{(i)}$, and the clean reference patches $\mathbf{y}^{(i)}$ to form the rows of the following matrices:

$$\hat{\mathbf{X}}_t = \underbrace{\begin{bmatrix} — (\hat{\mathbf{x}}_t^{(1)})^\top — \\ — (\hat{\mathbf{x}}_t^{(2)})^\top — \\ \vdots \\ — (\hat{\mathbf{x}}_t^{(N)})^\top — \end{bmatrix}}_{\in \mathbb{R}^{N \times d}}, \quad \mathbf{X}_t = \underbrace{\begin{bmatrix} — (\mathbf{x}_t^{(1)})^\top — \\ — (\mathbf{x}_t^{(2)})^\top — \\ \vdots \\ — (\mathbf{x}_t^{(N)})^\top — \end{bmatrix}}_{\in \mathbb{R}^{N \times d}}, \quad \mathbf{Y} = \underbrace{\begin{bmatrix} — (\mathbf{y}^{(1)})^\top — \\ — (\mathbf{y}^{(2)})^\top — \\ \vdots \\ — (\mathbf{y}^{(Y)})^\top — \end{bmatrix}}_{\in \mathbb{R}^{Y \times d}}. \tag{S26}$$

Then, we can write batched denoising in vectorized form as

$$\hat{\mathbf{X}}_t = \underbrace{\operatorname{softmax}\left(\mathbf{L}(\mathbf{X}_t, \mathbf{Y})\right)}_{\mathbf{W} \in \mathbb{R}^{N \times Y}} \mathbf{Y}, \tag{S27}$$

where $\mathbf{L}$ is a matrix in $\mathbb{R}^{N \times Y}$ whose (i,j)-th entry is $-\frac{1}{2\sigma_t^2} \left\| \mathbf{x}_t^{(i)} - \alpha_t \mathbf{y}^{(j)} \right\|_2^2$ and matrix $\mathbf{W} = \operatorname{softmax}(\mathbf{L})$ represents row-wise normalization of $\mathbf{L}$ so that each row sums to 1. In PyTorch, this can be implemented using `torch.cdist` or by forming $\mathbf{L}$ via matrix multiplications, applying a row-wise softmax, and finally computing the weighted sum of clean patches through the matrix product $\mathbf{WY}$.

Runtime. The asymptotic complexity of each denoising step is $\mathcal{O}(N \times Y \times d)$, which becomes $\mathcal{O}(N^2 \times d)$ when the number of clean patches and noisy patches is the same.

S3.2. Acceleration by a Fused FlashAttention-like Kernel

Connections of the closed-form denoiser to attention [68]. An important insight to our closed-form denoiser is that it can be treated as scaled dot-product attention. Thus, we can benefit from using FlashAttention and potentially any other hardware-accelerated algorithm. Observe that the Gaussian kernel in general can be decomposed as

$$k_\sigma(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|_2^2}{2\sigma^2}\right) = \exp\left(-\frac{\|\mathbf{x}\|_2^2 - 2\mathbf{x}^\top \mathbf{y} + \|\mathbf{y}\|_2^2}{2\sigma^2}\right) = \exp\left(-\frac{\|\mathbf{x}\|_2^2/2}{\sigma^2}\right) \exp\left(\frac{\mathbf{x}^\top \mathbf{y} - \|\mathbf{y}\|_2^2/2}{\sigma^2}\right). \tag{S28}$$

In particular, the (i,j)-th entry of $\mathbf{W}$ as softmax weight can be simplified as follows due to the shift-invariance of softmax:

$$w_{ij} = \frac{k_{\sigma_t}\left(\mathbf{x}_t^{(i)}, \alpha_t \mathbf{y}^{(j)}\right)}{\sum_{j'} k_{\sigma_t}\left(\mathbf{x}_t^{(i)}, \alpha_t \mathbf{y}^{(j')}\right)} \tag{S29}$$

$$= \frac{\exp\left(-\frac{\left\|\mathbf{x}_t^{(i)}\right\|_2^2/2}{\sigma_t^2}\right) \exp\left(\frac{\left(\mathbf{x}_t^{(i)}\right)^\top \left(\alpha_t \mathbf{y}^{(j)}\right) - \left\|\alpha_t \mathbf{y}^{(j)}\right\|_2^2/2}{\sigma_t^2}\right)}{\sum_{j'} \exp\left(-\frac{\left\|\mathbf{x}_t^{(i)}\right\|_2^2/2}{\sigma_t^2}\right) \exp\left(\frac{\left(\mathbf{x}_t^{(i)}\right)^\top \left(\alpha_t \mathbf{y}^{(j')}\right) - \left\|\alpha_t \mathbf{y}^{(j')}\right\|_2^2/2}{\sigma_t^2}\right)} \tag{S30}$$

$$= \frac{\exp\left(\frac{\left(\mathbf{x}_t^{(i)}\right)^\top \left(\alpha_t \mathbf{y}^{(j)}\right) - \left\|\alpha_t \mathbf{y}^{(j)}\right\|_2^2/2}{\sigma_t^2}\right)}{\sum_{j'} \exp\left(\frac{\left(\mathbf{x}_t^{(i)}\right)^\top \left(\alpha_t \mathbf{y}^{(j')}\right) - \left\|\alpha_t \mathbf{y}^{(j')}\right\|_2^2/2}{\sigma_t^2}\right)}. \tag{S31}$$

Now, using a homogeneous coordinate trick, we can express the noisy patch $\mathbf{x}_t$ and scaled clean patch $\alpha_t \mathbf{y}^{(j)}$ as a query and a key respectively so that it becomes a dot product inside the exponentials. Specifically, we denote

$$\mathbf{q}^{(i)} = \begin{bmatrix} \mathbf{x}_t^{(i)} \\ 1 \end{bmatrix}, \quad \mathbf{k}^{(j)} = \begin{bmatrix} \alpha_t \mathbf{y}^{(j)} \\ -\left\|\alpha_t \mathbf{y}^{(j)}\right\|_2^2 / 2 \end{bmatrix} \tag{S32}$$

both as $d+1$ dimensional vectors, and we have

$$\exp\left(\frac{\left(\mathbf{x}_t^{(i)}\right)^\top \left(\alpha_t \mathbf{y}^{(j)}\right) - \left\|\alpha_t \mathbf{y}^{(j)}\right\|_2^2 / 2}{\sigma_t^2}\right) = \exp\left(\frac{\left(\mathbf{q}^{(i)}\right)^\top \mathbf{k}^{(j)}}{\sigma_t^2}\right). \tag{S33}$$

This suggests that once we define two more stacked matrices,

$$\mathbf{Q} = \underbrace{\begin{bmatrix} — \, (\mathbf{q}^{(1)})^\top — \\ — \, (\mathbf{q}^{(2)})^\top — \\ \vdots \\ — \, (\mathbf{q}^{(N)})^\top — \end{bmatrix}}_{\in \mathbb{R}^{N \times (d+1)}} \quad \text{and} \quad \mathbf{K} = \underbrace{\begin{bmatrix} — \, (\mathbf{k}^{(1)})^\top — \\ — \, (\mathbf{k}^{(2)})^\top — \\ \vdots \\ — \, (\mathbf{k}^{(Y)})^\top — \end{bmatrix}}_{\in \mathbb{R}^{Y \times (d+1)}}, \tag{S34}$$

we can write the weight matrix as

$$\mathbf{W} = \mathrm{softmax}(\mathbf{L}(\mathbf{X}_t, \mathbf{Y})) = \mathrm{softmax}(\mathbf{Q}\mathbf{K}^\top / \sigma_t^2). \tag{S35}$$

Then, with a dummy "value" $\mathbf{V} := \mathbf{Y}$, we obtain

$$\hat{\mathbf{X}}_t = \mathrm{softmax}(\mathbf{L}(\mathbf{X}_t, \mathbf{Y}))\mathbf{Y} = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sigma_t^2}\right)\mathbf{V} \tag{S36}$$

which is exactly the scaled dot product used in attention, with scale $1/\sigma_t^2$ (instead of the originally used $1/\sqrt{d}$). We can therefore utilize acceleration techniques like FlashAttention for our closed-form denoiser.

FlashAttention [13, 14, 57]. We employ the FlashAttention family of kernels [13, 14, 57] to accelerate our patch denoising operations. FlashAttention restructures the standard scaled dot-product computation into SRAM-resident tiles and applies an online softmax update, eliminating the need to materialize the full attention matrix and thereby reducing High Bandwidth Memory (HBM) traffic and peak memory while maintaining exact outputs.

We use PyTorch's fused `scaled_dot_product_attention` operator with the *memory-efficient* backend, which dispatches to CUTLASS-based FlashAttention kernels. As these kernels only support mask-like bias formats, the formulation introduced in the previous section—where the per-vector offset is absorbed via a *homogeneous coordinate* trick—allows us to omit the bias tensor entirely and remain on the FlashAttention execution path. Passing a general dense bias would otherwise route the operator to a slower fallback implementation.

To satisfy kernel alignment constraints, we pad the patch-vector dimension to architecture-friendly multiples (e.g., 16/32/64). Padding with zeros leaves the computation unchanged while enabling more efficient vectorized memory access. We also observe that FlashAttention loses efficiency when the patch-vector dimension becomes large (typically $d \gtrsim 1000$), so we keep all patch-vector dimensions below this range to consistently benefit from the fused kernels.

Runtime with FlashAttention. The asymptotic bound is still $\mathcal{O}(N \times Y \times d)$, and $\mathcal{O}(N^2 \times d)$ when $N = Y$. Practically, however, the fused attention kernel provides a consistent $1.8\times$ speedup compared to the vanilla implementation across all image resolutions, see Table 2.

S3.3. Acceleration with an Image Auto-Encoder (VAE)

Motivation. As in large diffusion models, attention-like computations become prohibitively expensive for high-resolution images because the cost scales quadratically with sequence length. If we extract one patch per pixel (stride = 1), then the number of noisy queries N and clean keys Y are both proportional to the number of pixels, resulting in an $\mathcal{O}(NY)$ interaction that quickly dominates runtime. The latent diffusion literature alleviates this by spatially compressing the image using a VAE, reducing the sequence length while preserving perceptual quality with only mild compression loss.

Our VAE setup. We use the VAE from the FLUX model [39], which downsamples each spatial dimension by a factor of 8 and encodes images into a 16-channel latent image. This corresponds to a $1/64$ reduction in pixel count, and—when patches are extracted with stride 1—a $1/64$ reduction in the total number of patches for both noisy and clean images. For the same patch size, the number of channels—and hence the patch vector dimensionality—increases by a factor of $16/3 \approx 5.33$. However, to keep comparisons fair, we use a latent patch size of 7, giving a patch vector dimension

$$d = 7 \times 7 \times 16 = 784,$$

which is comparable to our canonical RGB patch dimension of $15 \times 15 \times 3 = 675$. Empirically, our single-scale and multi-scale image denoisers operate just as well on latent tensors of shape $(H/8) \times (W/8) \times 16$ as on RGB images of shape $H \times W \times 3$.

Runtime. The theoretical complexity of the denoising kernel remains $\mathcal{O}(N \times Y \times d)$, or $\mathcal{O}(N^2 d)$ in the common setting $Y = N$. However, the VAE provides a large constant-factor speedup. (1) The VAE encoder/decoder is convolutional and thus scales linearly with the number of patches, adding minimal overhead. (2) The denoising step now operates on only $(1/64)N$ noisy patches and $(1/64)Y$ clean patches, reducing the cost of the core batched patch denoising step by a factor of

$$\left(\frac{1}{64}\right)^2 = \frac{1}{4096}$$

relative to the RGB-space computation. As shown in Table 2, the latent-space version yields substantial acceleration, especially at high resolutions.

S3.4. Acceleration with Approximate Nearest Neighbours

The closed-form denoiser in Equation S19 can be further accelerated by restricting the kernel summation to the top-k approximate nearest neighbour (ANN) patches of each query. We adopt the Faiss [17, 33] implementation of the inverted file index [61] for ANN search.

Algorithm. The dataset of clean patches $\mathcal{Y}$ is first partitioned into n_{list} clusters using the k-means algorithm, and the resulting centroids define an inverted index. This index is built once and reused across all denoising steps. At inference time, for each noisy query patch $\mathbf{x}_t$ and each diffusion timestep, we retrieve its approximate k nearest neighbours by:

1. finding the closest n_{probe} centroids and their associated clusters;
2. performing an exact search for the k nearest neighbours within the union of those n_{probe} clusters.

The approximation comes from step (1): if we set $n_{\text{probe}} = n_{\text{list}}$, the method reduces to exact nearest neighbour search over all patches. In our experiments, we follow a standard choice and set $n_{\text{list}} = \sqrt{Y}$, where Y is the total number of clean patches.

We optionally apply product quantization (PQ) on top of the inverted file index. PQ compresses each d-dimensional patch vector into a short code (a few bytes), which reduces memory consumption and replaces expensive distance computations with table lookups, yielding an additional constant-factor speedup in both memory and runtime.

Runtime. For simplicity, we assume $Y = N$, i.e., the number of clean patches matches the number of noisy patches. With $n_{\text{list}} = \sqrt{Y} = \sqrt{N}$, each cluster contains approximately $N/n_{\text{list}} = \sqrt{N}$ patches on average. For a single query, the cost of: (1) scanning all $n_{\text{list}} = \sqrt{N}$ centroids to find the closest n_{probe} is $\mathcal{O}(\sqrt{N})$, and (2) searching within the n_{probe} selected clusters is $\mathcal{O}(n_{\text{probe}}\sqrt{N})$. Thus, each query costs $\mathcal{O}((n_{\text{probe}} + 1)\sqrt{N})$, and performing this search for all N noisy patches yields a total complexity of

$$\mathcal{O}\big(N(n_{\text{probe}} + 1)\sqrt{N}\big) = \mathcal{O}(N^{3/2})$$

(up to constant factors and the patch dimension d). This is an improvement over the $\mathcal{O}(N^2)$ cost of exact closed-form denoising, and in practice—together with PQ and GPU acceleration—provides substantial speedups while maintaining high-quality denoising.

S3.5. Implementation of High-Resolution Generation

We combine all acceleration techniques described above to enable gigapixel-scale generation with our method. We first encode each input image with the FLUX VAE and initialize the denoising process in the latent space. All denoising operations

are performed on latent tensors, and the final high-resolution result is obtained by decoding the generated latent back to RGB space.

For the weighted kernel summation, we use a FlashAttention-style implementation instead of naive PyTorch operations. Across image scales, we employ different denoiser types for a good trade-off of efficiency and quality: we use the exact closed-form denoiser on coarse scales whose spatial resolution is below 500×500 pixels, where exact computation is affordable; for all finer scales, we use approximate nearest neighbours (ANN) with product quantization (PQ). Our ANN configuration uses $n_{\text{probe}} = 40$, $k = 5$, and PQ codes of 28 bytes per patch vector, which significantly reduces memory footprint while maintaining fidelity. We run the sampler for $T = 20$ denoising steps with full stochasticity $\eta = 1.0$.

As shown in Figs. 5 and S9, our implementation produces high-quality gigapixel images on an NVIDIA RTX 6000 PRO GPU, with end-to-end runtimes well under an hour (the three examples shown take approximately 13, 33, and 39 minutes, respectively).

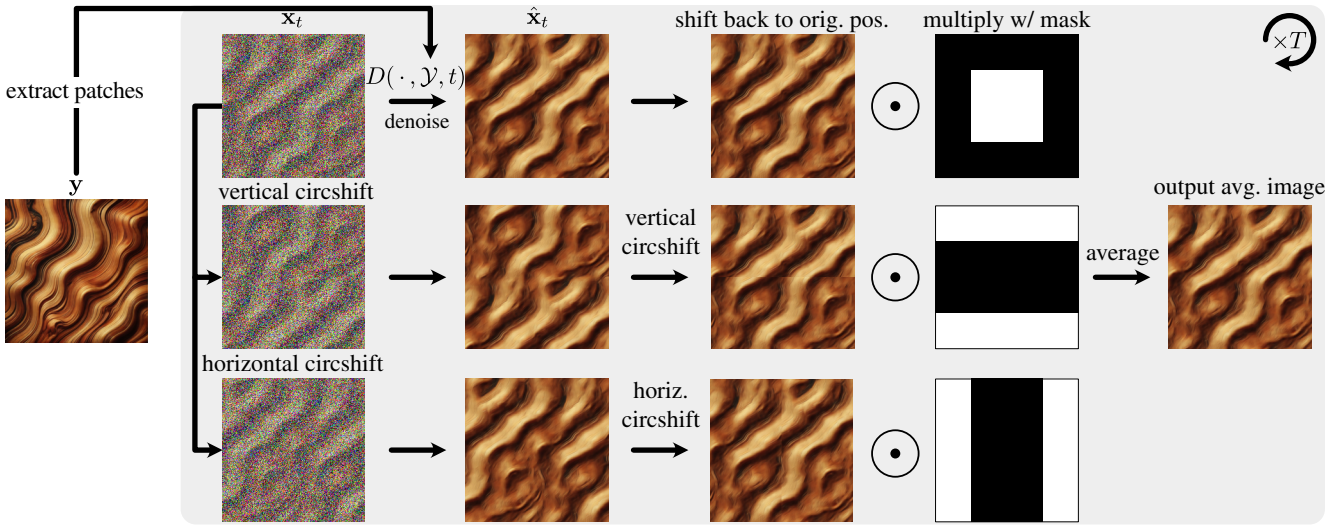


Figure S3. Overview of tileable image generation. Starting with a noisy image $\mathbf{x}_t$, we circularly shift ("circshift") this image vertically and horizontally, denoise the resulting images, and shift them back. Then, we multiply each image with a mask and average the resulting images to compute an output image. This procedure is repeated iteratively through the reverse diffusion process with the coarse-to-fine image sampling to generate tileable images.

Algorithm S1 Single-Scale Inversion Sampling

procedure SAMPLEIMAGEINVERSION($\mathbf{y}, t'$) ▷ t' is the target inversion timestep
  $\mathcal{Y} = \{\mathbf{P}^{(1)}\mathbf{y}, \ldots, \mathbf{P}^{(N)}\mathbf{y}\}$ ▷ extract clean reference patches
  $\mathbf{x}_1 \leftarrow \alpha(1)\mathbf{y} + \sigma(1)\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ first step warmup
  for $t = 1, \ldots, t' - 1$ **do** ▷ clean → noisy diffusion steps
    $\hat{\mathbf{x}}_t \leftarrow \text{IMGDENOISE}(\mathbf{x}_t, \mathcal{Y}, t)$
    $\hat{\boldsymbol{\epsilon}}_t \leftarrow \big(\mathbf{x}_t - \alpha(t)\,\hat{\mathbf{x}}_t\big)/\sigma(t)$
    $\mathbf{x}_{t+1} \leftarrow \alpha(t+1)\,\hat{\mathbf{x}}_t + \sigma(t+1)\,\hat{\boldsymbol{\epsilon}}_t$ ▷ DDIM inversion step with $\eta(t) = 0$
  end for
  return $\mathbf{x}_{t'}$
end procedure

S4. Application Details

S4.1. Tileable Image Generation

We illustrate the procedure for tileable image generation in Figure S3.

S4.2. Single-Scale Inversion Algorithm

DDIM inversion with warm-start. To obtain a noisy version of an input image $\mathbf{y}$ at a target diffusion timestep t', we use a deterministic DDIM-style inversion. The procedure, shown in Algorithm S1, follows the standard DDIM update in the forward direction (i.e., the ODE view with $\eta(t) = 0$), but begins with a small warm-start around the clean image. Specifically, we add a small noise at the first timestep using $(\alpha(1), \sigma(1))$ to obtain $\mathbf{x}_1$, and then apply the usual DDIM update from $t = 1$ up to $t = t'$. Here, we can invert the clean image all the way to $t' = T$.

This warm-start is a lightweight heuristic that avoids the trivial fixed point obtained when our closed-form denoiser is applied directly to $\mathbf{y}$. Apart from this first step, the trajectory is identical to standard DDIM inversion. Throughout the paper, we refer to this procedure simply as "DDIM inversion" for brevity.

Optional stopping at intermediate noise levels. While conventional DDIM inversion typically runs to the maximal noise level $\sigma(T) \approx 1$, our method allows stopping at any intermediate timestep t', yielding $\mathbf{x}_{t'}$ with noise level $\sigma(t')$. This provides a convenient knob that controls how much structure is preserved before the (guided) reverse process is applied. Lower t' retains more image detail, while higher t' yields stronger edits, making this flexibility useful for style transfer and structural-analogy tasks.

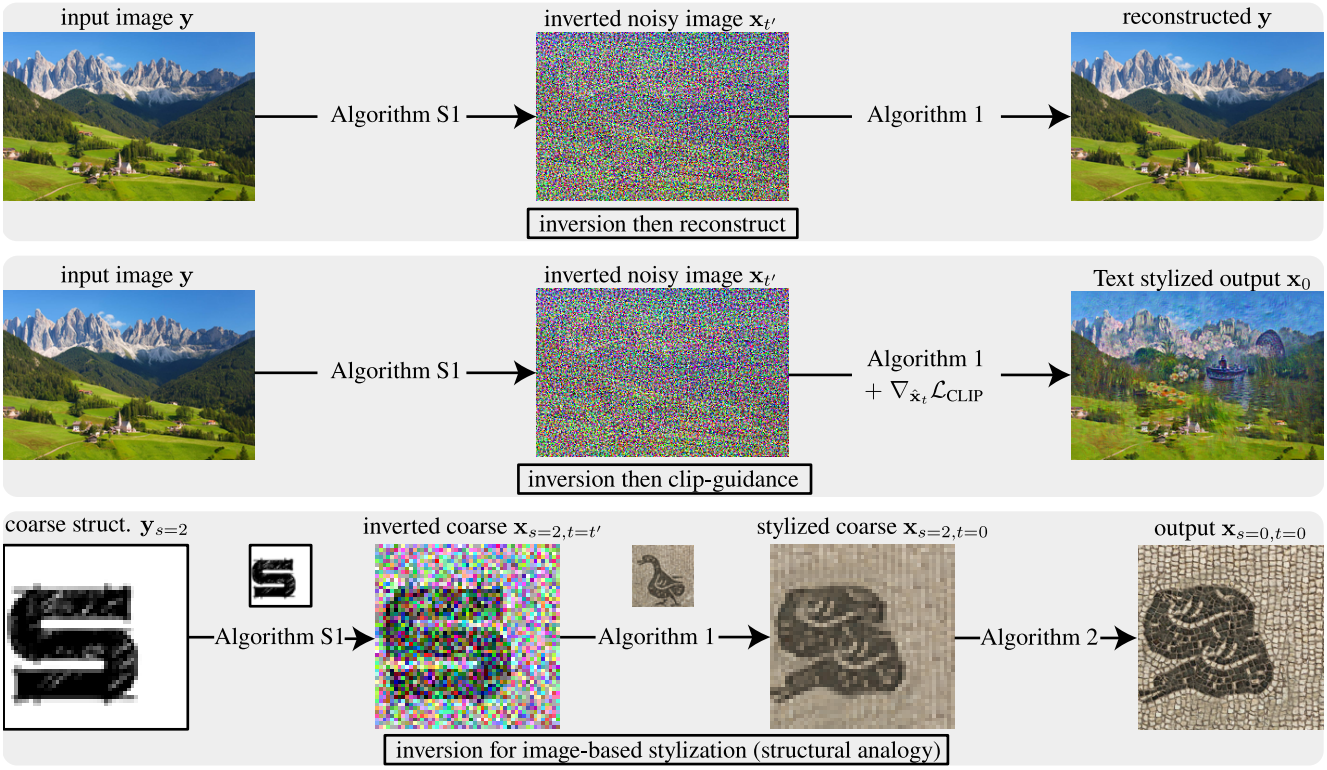


Figure S4. **Overview of inversion algorithms across different use cases. Top row:** Using our inversion method (Algorithm S1), an input image can be inverted to a noise level t' and then reconstructed using the same patch dataset via our single-scale denoiser (Algorithm 1). **Middle row:** For text stylization, we apply the same inversion procedure to obtain the noisy image, but during denoising we additionally incorporate CLIP-guided gradients to produce a "Monet"-style output. **Bottom row:** Structural analogy. We first invert the coarse scale $s = 2$ of the structure image using its own patch dataset to obtain a coarse noisy image at t'. We then *denoise this inverted coarse image using the style image's patch dataset* (note the patch dataset change at this stage) and finally apply coarse-to-fine sampling to produce the full-resolution output (Algorithm 2).

Properties and applications. For any inversion depth t', running our sampler (Algorithm 1) from t' back to $t = 0$ with the same patch dataset reconstructs the original image up to small numerical differences. The usefulness of inversion emerges when the reverse denoising is *not* a simple reconstruction but a guided or cross-image process. Because the inversion step preserves the coarse structure of the input image in $\mathbf{x}_{t'}$, the subsequent guided reverse pass can introduce controlled edits while still respecting the original layout. For example, replacing the reverse denoiser with a CLIP-guided version enables text-driven stylization, where global structure is preserved and local texture is modified. Similarly, in structural-analogy experiments, we invert an image using patches from a source and denoise using patches from a target, retaining the source layout while transferring target appearance. See Figure S4 for illustration of these three use cases.

S4.3. Structural Analogy

We illustrate our structural-analogy procedure in the bottom row of Figure S4. In this setting, inversion (Algorithm S1) is performed using the patch dataset extracted from the *structure* image, producing a coarse noisy image at noise level t'. During denoising (Algorithm 1), we then *switch to the patch dataset of the style image* to transfer its local appearance. Finally, we apply coarse-to-fine sampling to obtain the full-resolution output (Algorithm 2).

S4.4. Text-Based Style Transfer

The CLIP guidance used in text-based style transfer incorporates the following image and text augmentation used in Text2LIVE [2] and SinDDM [38], which we include here for completeness. Specifically, we compute CLIP text embeddings for each of the text prompts below (where "{}" denotes the prompt input), and compute the CLIP image embedding after applying each of the image augmentations below to the input image. The CLIP loss is computed as the average cosine distance between these embeddings. We use a patch size of three by three pixels in the diffusion process; empirically, we

find that this patch size best captures textures and fine-scale image features, and the qualitative results align well with input text prompts.

Text augmentations.
- "photo of {}.",
- "high quality photo of {}.",
- "a photo of {}.",
- "the photo of {}.",
- "image of {}.",
- "an image of {}.",
- "high quality image of {}.",
- "a high quality image of {}.",
- "the {}.",
- "a {}.",
- "{}.",
- "{}",
- "{}!",
- "{}...",

Image augmentations.
- Random spatial crops: 0.85 and 0.95 of the image size.
- Random scaling: aspect ratio-preserving scaling, of both spatial dimensions.
- Multiplication by a random factor, sampled uniformly from the range [0.8, 1.2].
- Random horizontal-flipping is applied with probability p=0.5.
- Random color-jittering: we jitter the global brightness, contrast, saturation and hue of the image.

Sampling. We use single-scale sampling for this task. We first run the inversion algorithm (Algorithm S1) to map the clean input $\mathbf{y}$ to an inverted noisy image $\mathbf{x}_{t'}$ $(0 \to t')$. During the denoising phase $(t' \to 0)$, we largely follow SinDDM [38] for incorporating CLIP guidance, with some modifications. At each timestep we compute the denoised prediction $\hat{\mathbf{x}}_t$ and apply the CLIP-guided update

$$\hat{\mathbf{x}}_{t,\text{CLIP}} \leftarrow \gamma \nabla_{\hat{\mathbf{x}}_t} \mathcal{L}_{\text{CLIP}} + \lambda \hat{\mathbf{x}}_t + (1-\lambda)\hat{\mathbf{x}}_{t+1,\text{CLIP}}, \tag{S37}$$

where γ controls CLIP guidance strength and λ controls momentum (i.e., retention of the previous estimate). We use a time-dependent guidance schedule $\gamma(t) = \frac{\sigma(t)}{\sigma(t')}\gamma(t') \in [0, \gamma(t')]$, where $\gamma(t') = \gamma_{\text{max}}$ and the strength decays to 0 as t approaches 0. The momentum term helps prevent CLIP updates from being overridden by the denoiser [38]; we set $\lambda = 0.1$ for all experiments. Also, following [38], we do not use the raw CLIP gradient but instead compute

$$\mathbf{g}_t \leftarrow \nabla_{\lambda\hat{\mathbf{x}}_t + (1-\lambda)\hat{\mathbf{x}}_{t+1,\text{CLIP}}} \mathcal{L}_{\text{CLIP}}, \tag{S38}$$

$$\mathbf{m}_t \leftarrow \mathbf{1}\big\{\text{pixel-norm}(\mathbf{g}_t) > (1-f)\text{-quantile}\big\}, \tag{S39}$$

$$\mathbf{g}_t \leftarrow \|\hat{\mathbf{x}}_t \odot \mathbf{m}_t\|_2 \frac{\mathbf{g}_t \odot \mathbf{m}_t}{\|\mathbf{g}_t \odot \mathbf{m}_t\|_2}, \tag{S40}$$

where the *fill factor* $f \in [0,1]$ specifies the fraction of pixels allowed to be modified by the CLIP gradient at each step. The final normalization rescales the gradient so that its energy (within the modified region) matches that of the denoised prediction.

We use a patch size of 7 in all experiments. A sweep over $\sigma(t')$ is shown in Figure S5, demonstrating that style transfer is achievable across all inversion depths. A sweep over γ_{max} and f (Figure S6) shows that larger values of both parameters produce samples that more strongly follow the prompt. For comparison with SinDDM in Figure S16, we set $\gamma_{\text{max}} = 1.0$ and $f = 0.5$.

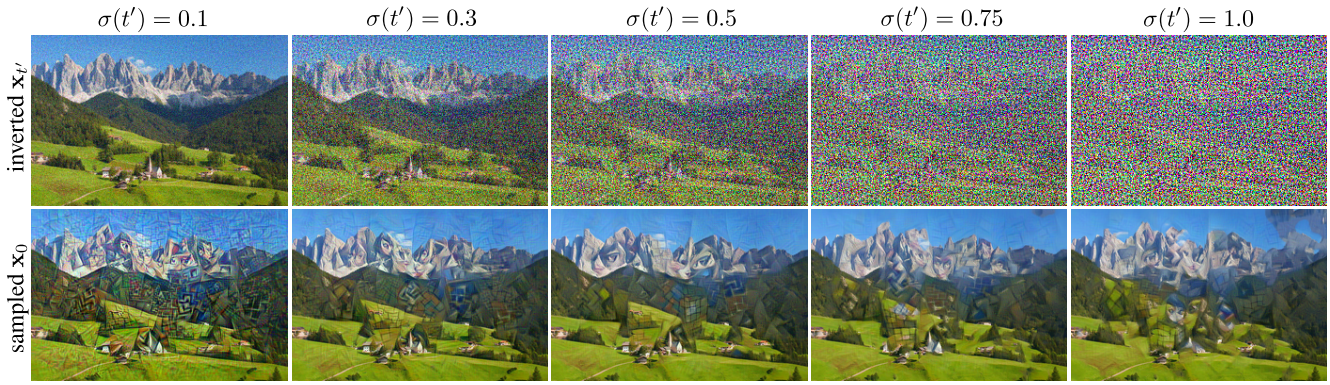


Figure S5. A sweep over inversion noise levels $\sigma(t') \in \{0.1, 0.3, 0.5, 0.75, 1.0\}$ using the prompt "Cubism." The top row shows the inverted images $\mathbf{x}_{t'}$ obtained from the clean input $\mathbf{y}$ ($0 \to t'$), and the bottom row shows the corresponding CLIP-guided denoised samples ($t' \to 0$). We fix $\gamma(t') = \gamma_{\text{max}} = 0.5$ and $f = 1.0$, and we adjust the number of timesteps so that the change in noise level $\sigma(t) - \sigma(t-1)$ is constant; thus higher $\sigma(t')$ requires more inversion and denoising steps (e.g., $\sigma(t') = 1.0$ corresponds to $t' = T = 200$ steps). Across all settings, in particular for $\sigma(t') = 1.0$, the input structure is preserved due to the inversion algorithm. Larger $\sigma(t')$ values (i.e., deeper inversion) reduce CLIP-induced noise in the final output, but also yield slightly weaker adherence to the original image.

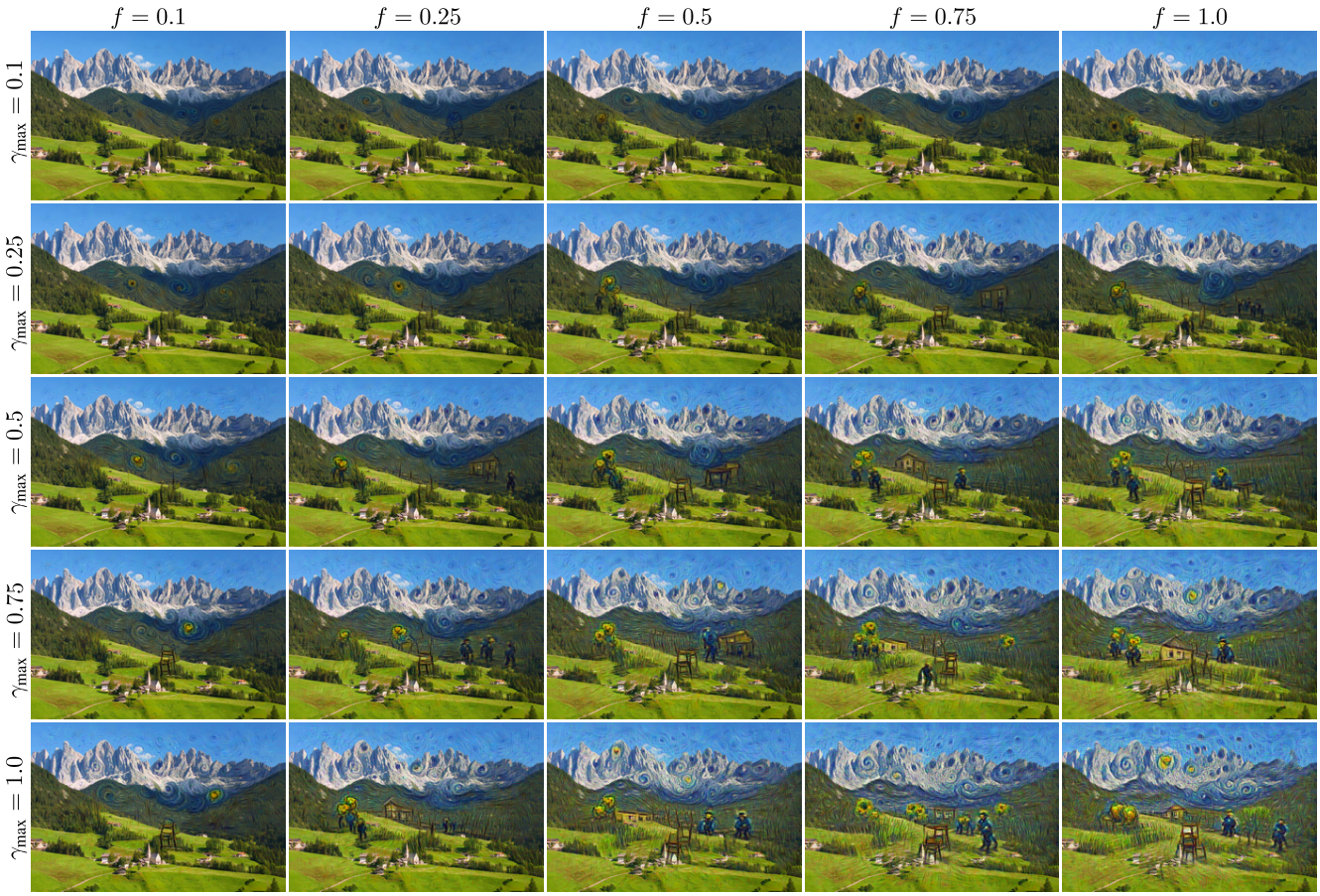


Figure S6. A sweep over guidance strength γ_{max} and fill factor f. All generations start from an inverted image at noise level $\sigma(t') = 0.3$ and use 60 inversion and 60 denoising steps. All generation uses prompt "Van Gogh". Stronger guidance yields more pronounced features aligned with the prompt, while larger fill factors cause edits to affect a greater portion of the image.

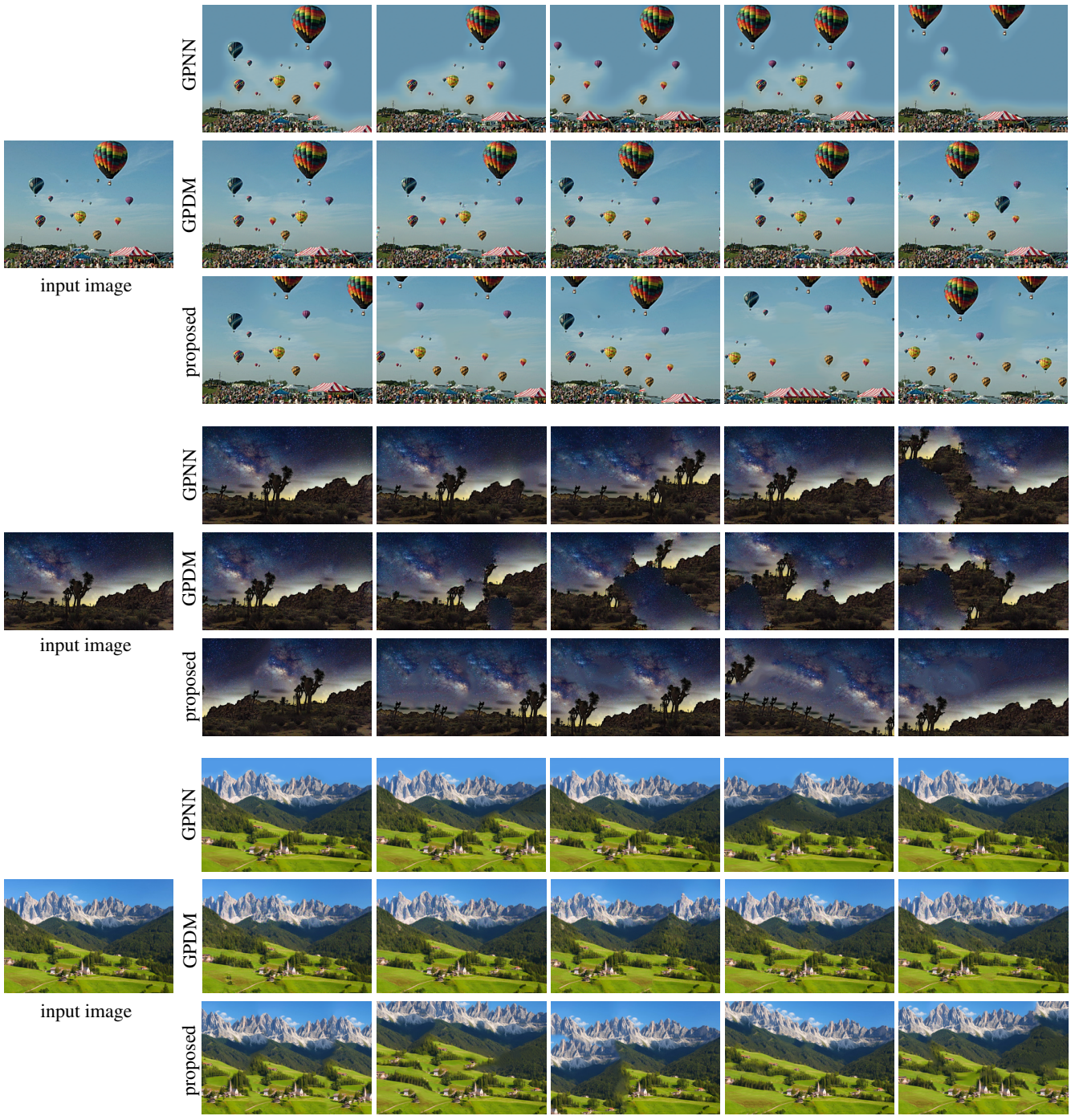


Figure S7. Randomly sampled outputs from GPNN, GPDM, and our proposed method. GPNN and GPDM outputs are often nearly identical to the input (also see Table S1).

S5. Supplementary Results

Unconditional generation. The 15 images used for unconditional generation were downloaded from the SinDDM repository, with some originating from Places50 and SIGD16. We compare against GPNN and GPDM in Figure S7. While their samples often closely resemble the input image, our method produces more diverse outputs. We quantify this using the near-duplicate ratio (Table S1). For each of the 15 input images, we sample 50 outputs for each method and report the fraction that

Method	r=100%	r=95%	r=90%	r=85%	r=80%	r=75%	r=70%
GPNN	0.13%	10.27%	16.93%	25.60%	34.80%	43.60%	52.40%
GPDM	0.00%	8.27%	12.80%	21.20%	25.07%	32.67%	44.27%
proposed ($T = 10, \eta = 0.0$)	0.00%	0.00%	0.00%	0.13%	0.40%	0.67%	2.13%

Table S1. Near-duplicate rates at different r%-pixel thresholds, with average RGB difference $<15/255$ defining a near-duplicate.

closely match the input. An output is considered a near-duplicate if at least $r\%$ of pixels have average RGB values within $15/255$ of the input.

Additional comparisons of unconditional generation with SinDDM [38] and the proposed method are shown in Figure S8. Our approach demonstrates similar quality with appreciably more diversity than the SinDDM results, which agrees with the quantitative results shown in the main paper.

Gigapixel generation. We show additional examples of gigapixel image generation in Figure S9. We generate these images in 33 minutes (*Moon*) and 39 minutes (*Tokyo*) on an NVIDIA RTX 6000 PRO by incorporating latent space diffusion, approximate nearest neighbors, and a fused attention operation.

Image retargeting. We show a comparison between our method and GPNN in Figure S10. Additional image retargeting examples are shown in Figure S11. Our method produces high-quality outputs across a variety of input images and aspect ratios.

Image symmetrization. Additional generated samples with vertical image symmetry are shown in Figure S12. We also provide additional generated tileable images in Figure S13.

Structural analogies. Figure S14 presents further structural analogy results alongside those produced by GPNN. The proposed approach offers similar qualitative performance to GPNN.

Comparison to large diffusion model. We compare our method against a large diffusion model, Nano Banana Pro [25] (see Figure S15). Because our approach generates output patches derived *exclusively* from the input image, it provides a clear provenance chain between input and output—a guarantee that large models cannot offer. Consequently, when Nano Banana Pro is applied to structural analogies, symmetrization, or tiling, its outputs fail to preserve the input patch distribution and do not satisfy the task constraints.

Text-based style transfer. Additional text-based style transfer examples are provided in Figure S16, together with outputs from SinDDM. Our method achieves comparable visual quality across a range of prompts spanning different artists and styles.

Region of interest (ROI)-conditioned generation. After each denoising step we apply a mask to keep the provided ROI fixed, and retain the denoising prediction in the unmasked regions. We proceed in this fashion for all stages of the coarse-to-fine image sampling procedure except for the finest stage to avoid seams. Example results are shown in Figure S17.

Visualization of coarse-to-fine image sampling. We visualize the coarse-to-fine image sampling process in Figure S18. The process is generated for $T = 100$ and $S = 4$, and we show images of $\mathbf{x}_{s,t}$, $\hat{\mathbf{x}}_{s,t}$, and $\tilde{\mathbf{x}}_{s,t}$ at $t = \{99, 80, 60, 40, 20, 0\}$ for all scales $s = \{3, 2, 1, 0\}$.

Patch sampling analysis. Given that at the final diffusion step ($\sigma(t) \to 0$), our method selects one patch from the dataset $\mathcal{Y}$ for each patch location in the sampled image, we can analyze the patch sampling distribution by counting the number of times each patch in $\mathcal{Y}$ is selected across multiple samples. Figure S19 shows histograms of the mean number of patch occurrences across 50 samples, overlaid on the input pyramid image (visualized at patch centers; borders are therefore zero). The entropy value measures the uniformity of patch sampling, with higher values indicating more uniform sampling of patches (with a maximum of 1.0, corresponding to perfectly uniform sampling). Increasing T and η increases patch uniformity (entropy H), but also increases similarity to the input image. For small T and η, we hypothesize that image reconstruction from patches (Algorithm 1, L14) constrains the layout at the coarsest scale, causing certain patches to be selected more frequently.

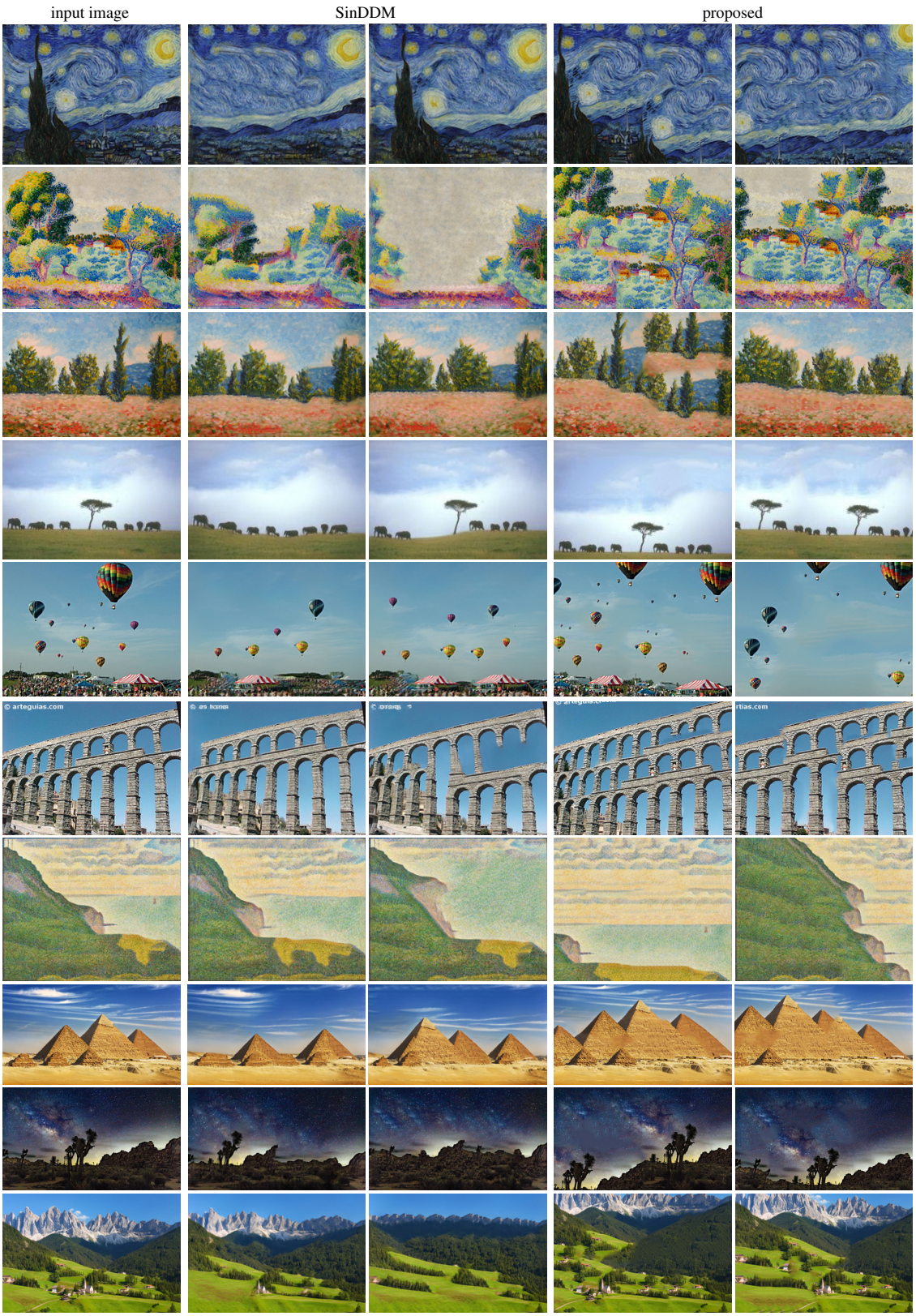


Figure S8. Additional comparisons to unconditional generation with SinDDM [38] and the proposed method. Samples from the proposed method show similar quality and slightly greater diversity than the outputs of SinDDM, which agrees with the quantitative results shown in the main paper.

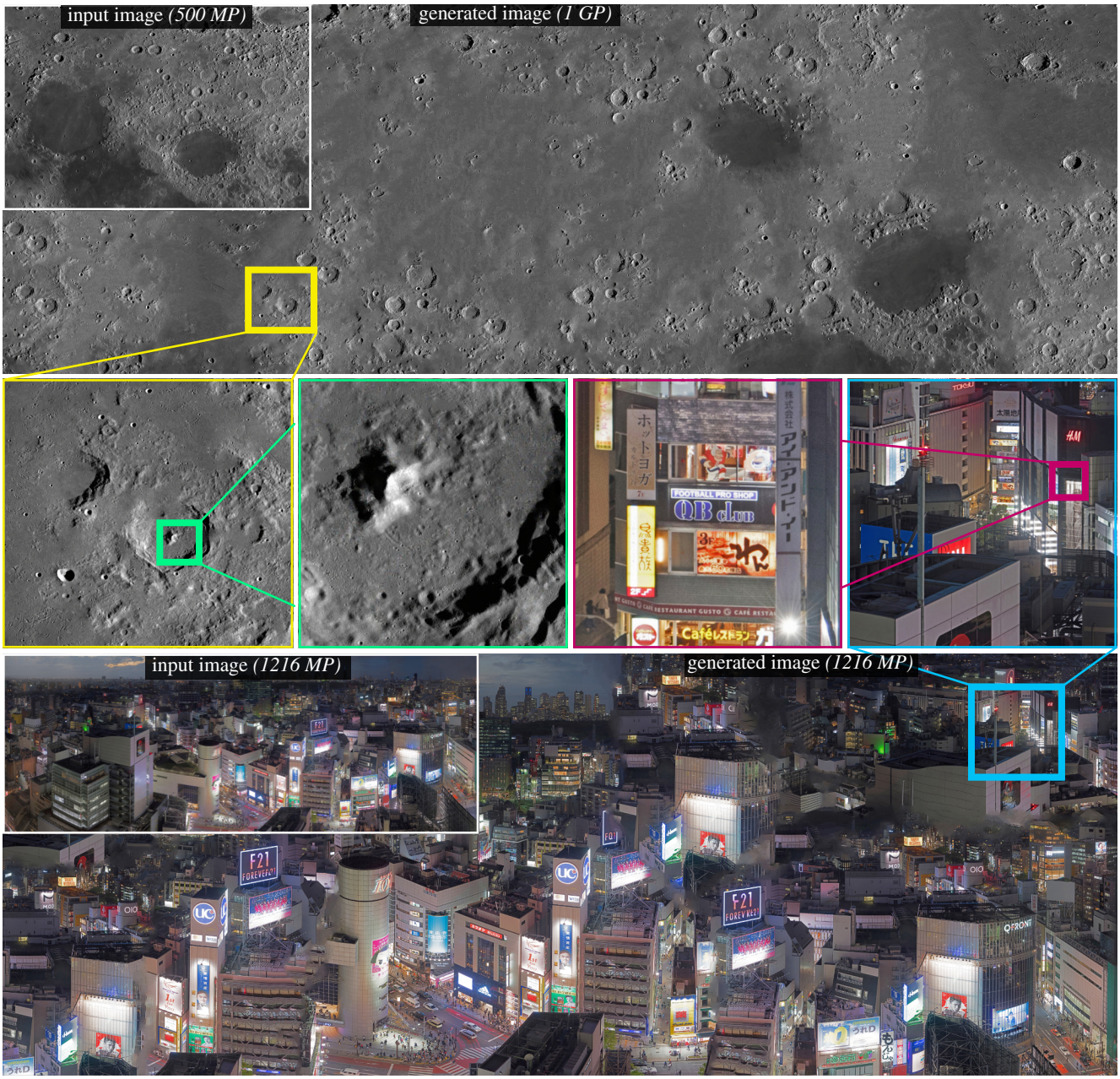


Figure S9. Supplemental gigapixel generation results. We use a combination of latent space diffusion, approximate nearest neighbors, and a fused attention kernel to generate these images in 33 minutes (top; *Moon*) and 39 minutes (bottom; *Tokyo*). *Image credits: top — NASA (public domain); bottom — Trevor Dobson (CC BY–NC–ND).*

Figure S10. Image retargeting. We show the input image (leftmost column) the output of GPNN [26] (middle column), and the results of the proposed method (right column). The proposed method achieves results of similar quality to GPNN.

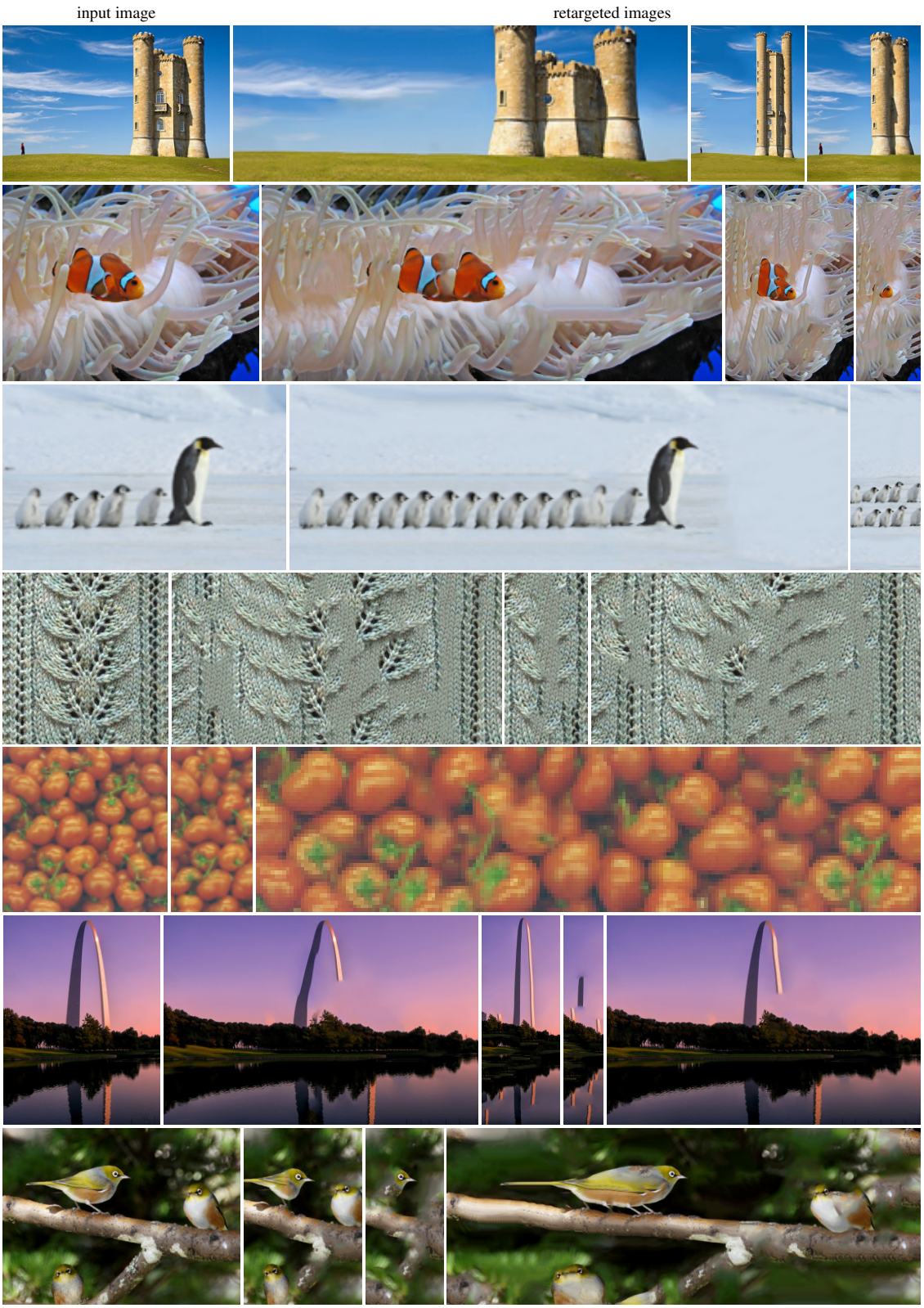


Figure S11. Supplementary results showing image retargeting with the proposed method. The input image is shown in the leftmost column of each row, and the other columns show retargeted samples.

Figure S12. Supplementary results showing generated image samples with vertical symmetry.

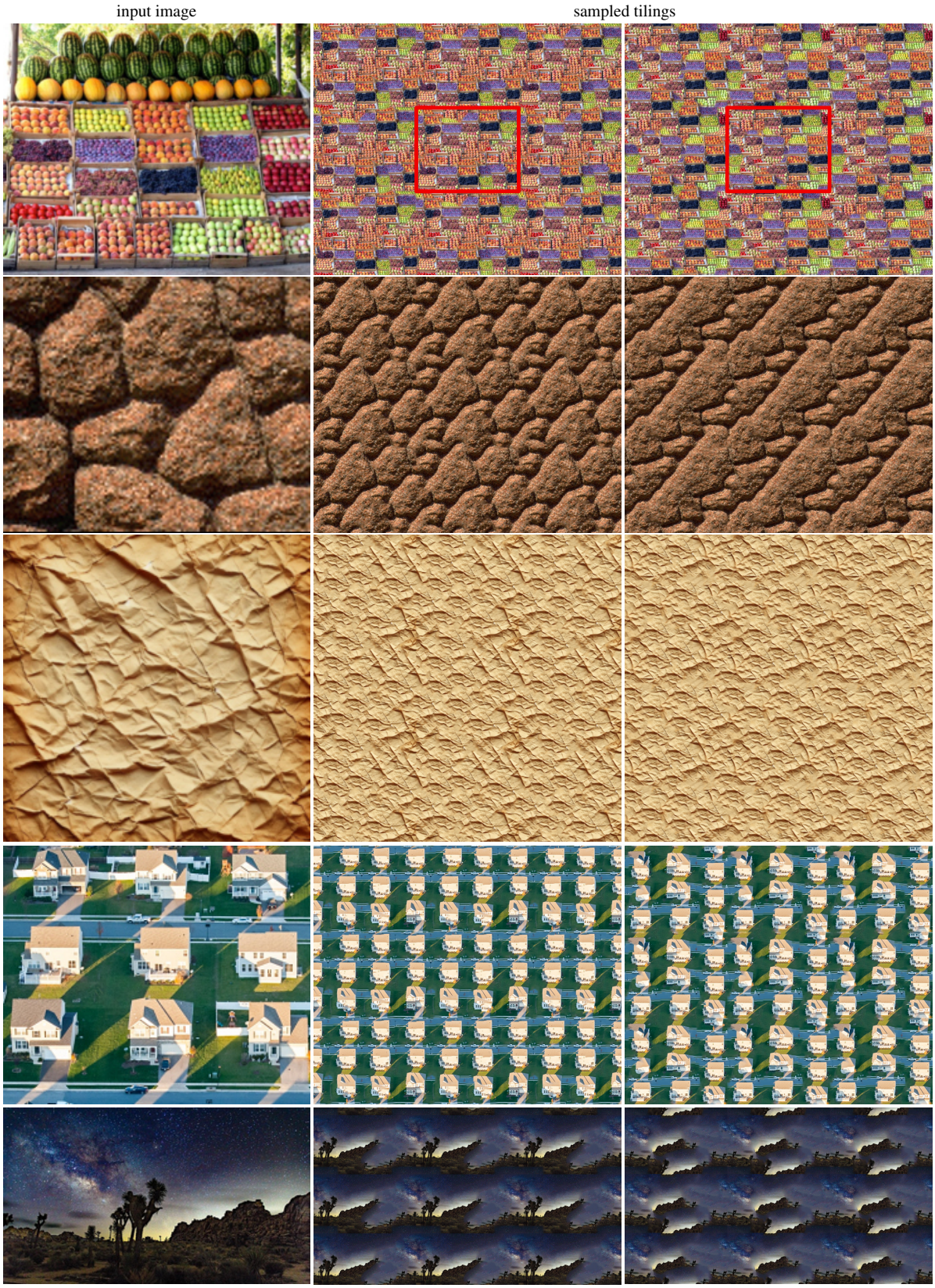


Figure S13. Supplementary results showing generated tileable image samples.

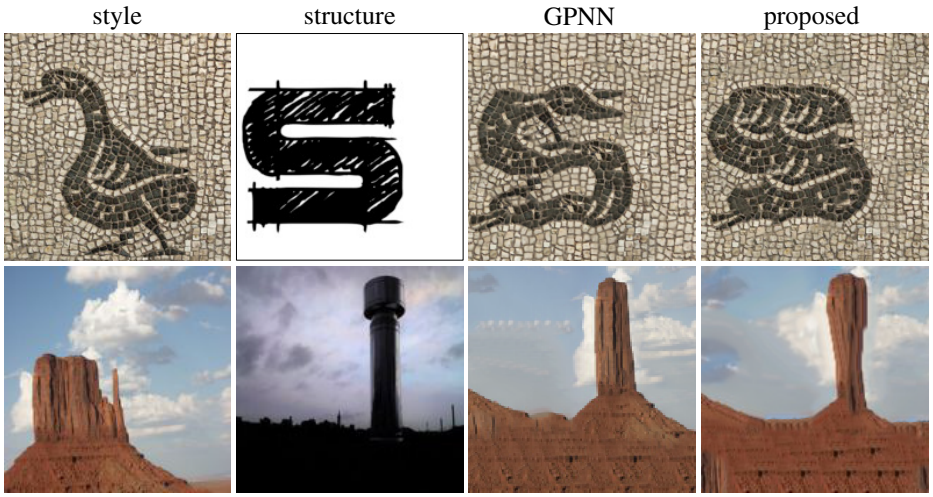
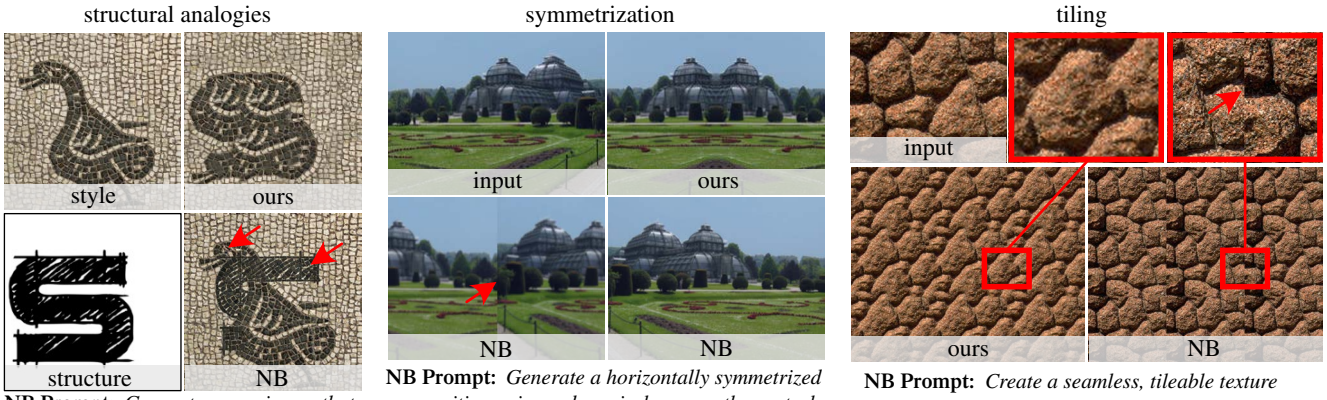


Figure S14. Supplementary results showing structural-analogies compared to GPNN. The proposed approach demonstrates qualitatively similar performance compared to GPNN.



NB Prompt: *Generate a new image that follows the exact structural layout and composition of structure.png, but applies the artistic style, color palette, and textures found in style.png.*

NB Prompt: *Generate a horizontally symmetrized composition, mirrored precisely across the central vertical axis. The structural layout, shapes, and local texture details must be wholly derived from the provided reference image. The center line where the two halves meet must be seamless and clean.*

NB Prompt: *Create a seamless, tileable texture map featuring a new arrangement of the patches found in the reference image. The output must be a repeatable pattern that aligns perfectly on all four sides with no visible seams.*

Figure S15. Nano Banana Pro (NB) results: outputs do not preserve the structure/input patch distribution (left), maintain symmetry (middle), or achieve seamless tiling (right).

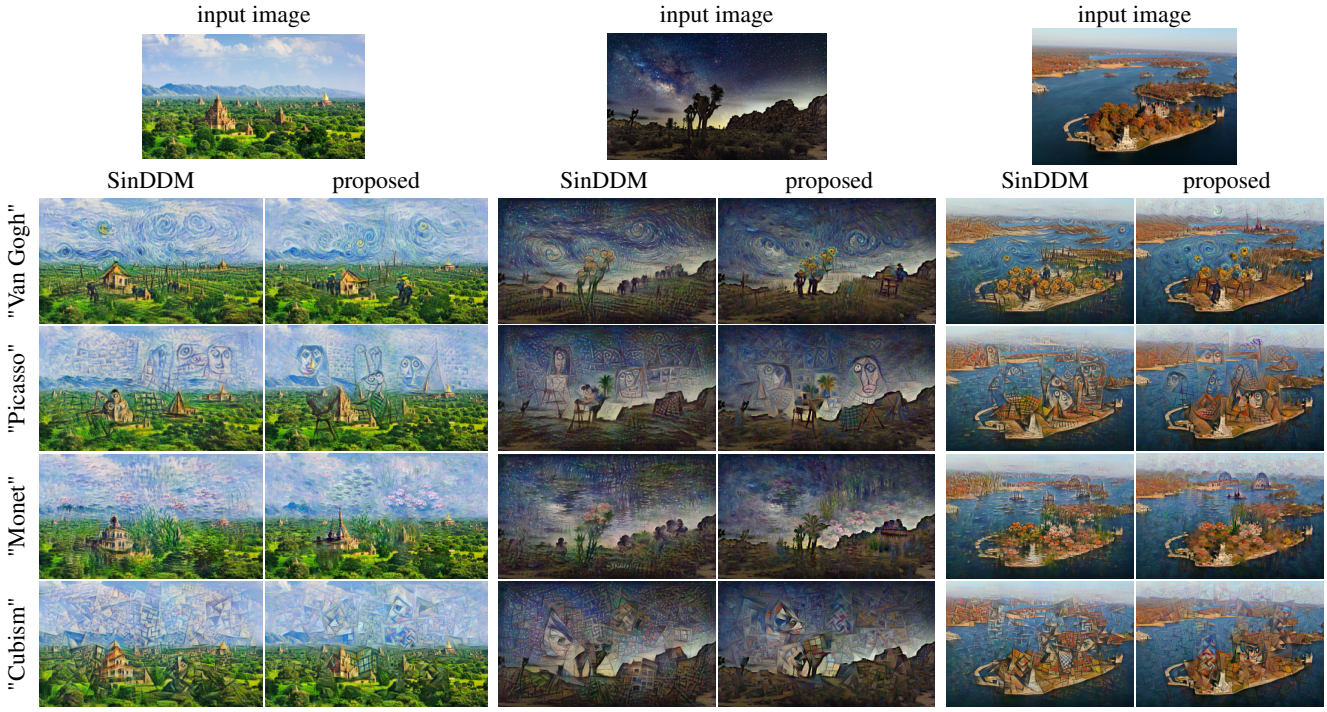


Figure S16. Supplementary results showing text-based style transfer compared to SinDDM. Patch size of 7×7, guidance of $\gamma_{\text{max}} = 1.0$, and fill factor $f = 0.5$ are used for the proposed. The proposed approach demonstrates qualitatively similar performance compared to SinDDM for a range of prompts corresponding to different artists and styles.

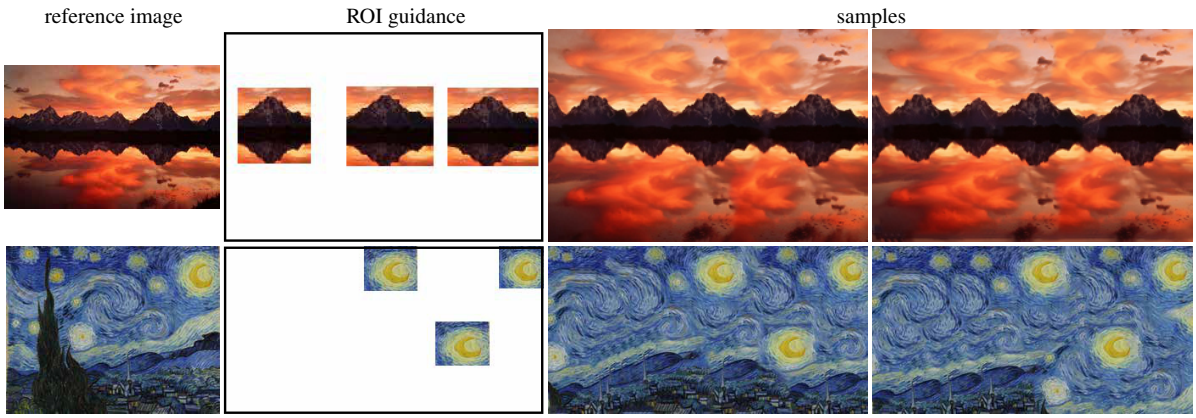


Figure S17. Supplementary results showing ROI-conditioned generation.

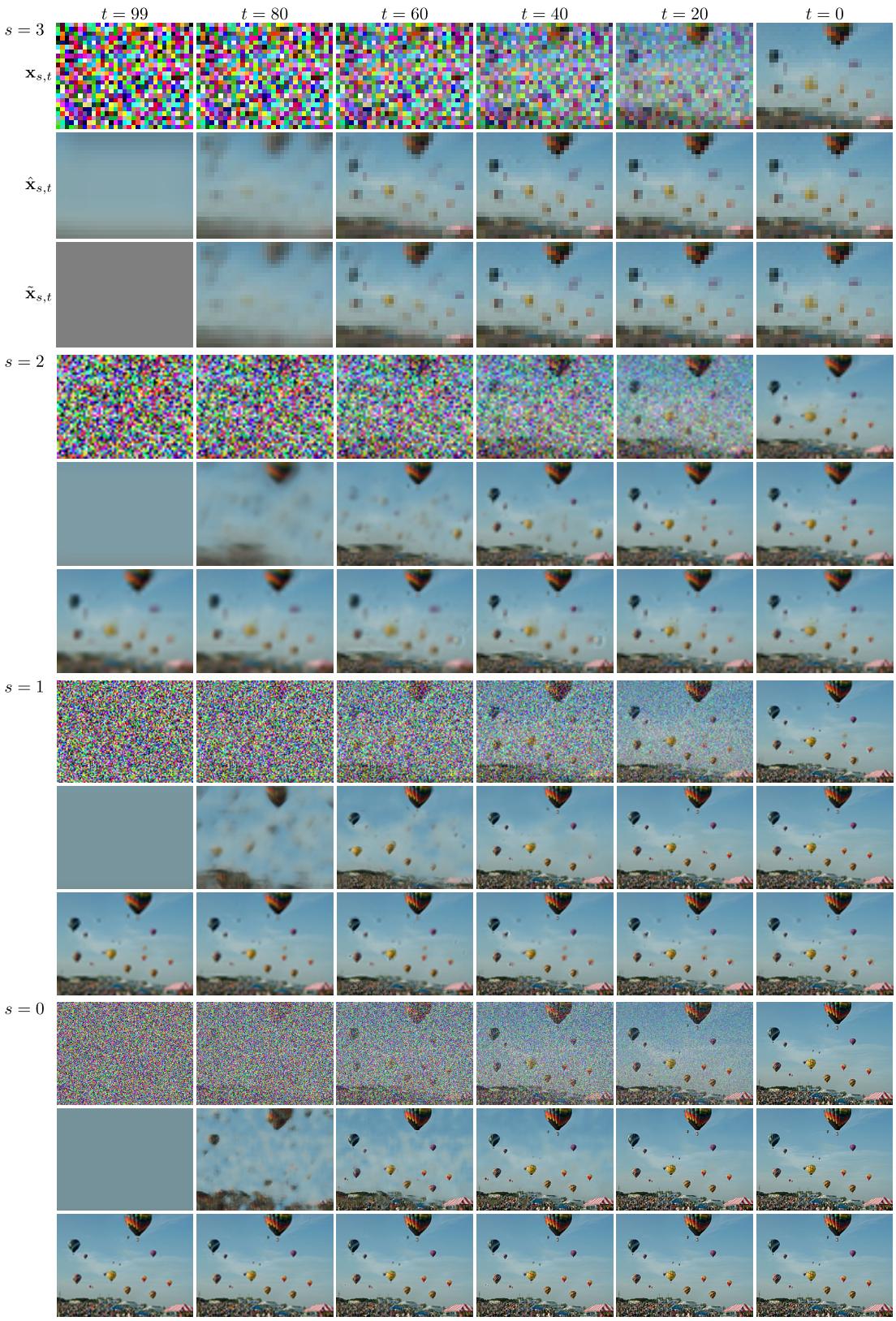


Figure S18. Visualization of coarse-to-fine image sampling for $S = 4$ and $T = 100$.

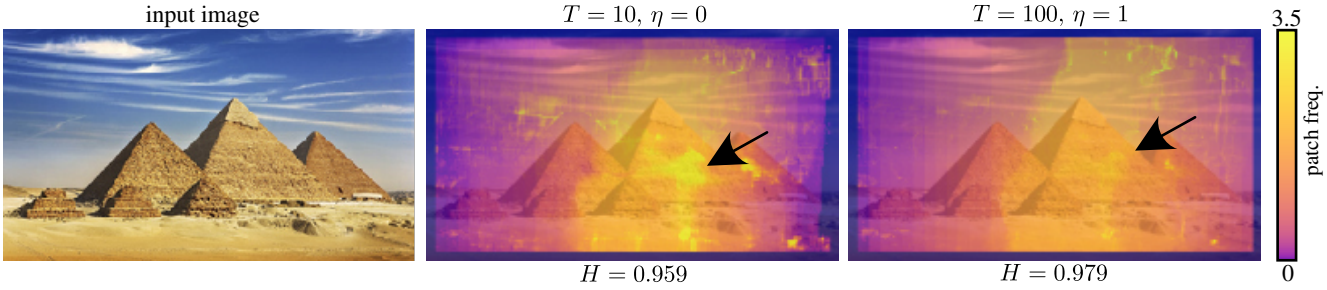


Figure S19. Patch frequency histogram & entropy (H) for different T, η for the unconditional generation. Increasing T and η increases patch uniformity/entropy (H).

S6. Discussions and Extensions

S6.1. Coarse-to-fine Sampling and Patch Size

While our approach uses explicit multi-scale modeling, the diffusion process itself can be viewed as operating in a multi-scale fashion as noise is gradually added or removed [1, 16, 30, 48, 54]. Hence, one open question is whether our explicit multi-scale approach can be modified, or even removed entirely and instead accommodated within the diffusion process.

One straightforward modification of our approach is to exchange the scale–noise inner and outer loops in Algorithm 2, i.e., first sample all scales at the first diffusion step, then move to the second diffusion step and sample all scales, and so on. This is largely equivalent to using multiple patch sizes to denoise at each noise level. We found that such unconditional generation samples have similar quality to those of our proposed algorithm, and intuitively, denoising steps at finer scales and higher noise levels (small s and large t) contribute little to the final sample.

Another alternative is to schedule the patch size during the diffusion process [49]. However, in the initial few steps, the patch size is on the order of the full image resolution, which removes the efficiency benefits of using a constant small patch size across scales, as in our design. Alternatively, approaches such as the recent Scale-Space Diffusion model [48], which incorporates explicit downsampling schedules, could be integrated with our method. We leave these directions for future work.

S6.2. Potential Extensions

There are several interesting extensions to our method, which we briefly discuss below.

Multiple images as references. One can easily extend our method to take multiple images as input by simply concatenating the patch datasets of all input images into one patch dataset $\mathcal{Y}$. This enables generation that draws from multiple sources, e.g., blending textures across images or aggregating complementary views of the same scene.

Priors in data-scarce or out-of-distribution domains. Single-image and patch-based models could provide useful priors when datasets are limited or the target domain differs substantially from large-model training data (e.g., astronomy, hyperspectral imaging, radar). These priors could also aid inverse problems, e.g., via Diffusion Posterior Sampling [11]. In this setting, the image sample is updated by alternating between an image prior (in this case, our patch-based image denoiser could potentially be used instead of a trained large diffusion denoiser) and a likelihood step that constrains the sample to be consistent with the observed measurements.

Complementarity with large diffusion priors. We envision that one could combine latent multi-image patch-based models with large-scale latent diffusion models by incorporating patch-based image priors as an additional guidance signal alongside standard text conditioning. This formulation would leverage the complementary strengths of large diffusion models—such as semantic understanding and global coherence—and patch-based models, which capture fine-scale image statistics from the input images. By adjusting the relative influence of these guidance signals, the generation could be steered to reflect both global semantics and local image structure. We see this as a particularly exciting future direction.