UltraFlash: Accelerating Megapixel Visual Synthesis

Phuc Lai[⋆] **Anh Nguyen**[⋆] **Phong Nguyen** **Anh Tran**
Qualcomm AI Research[†]

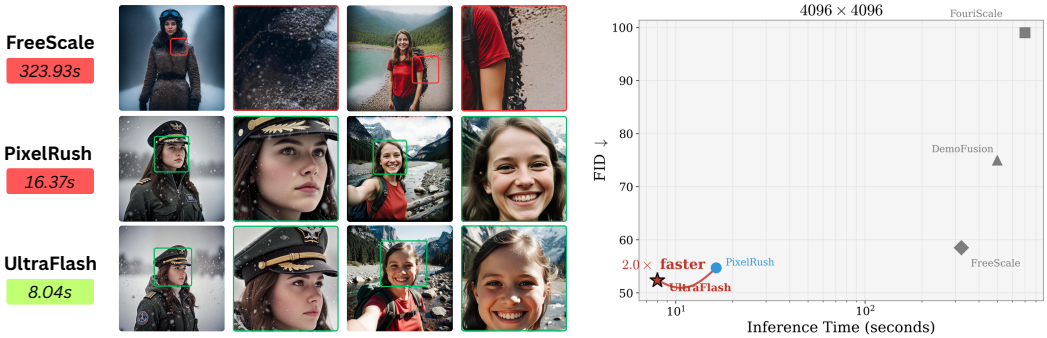


Figure 1: **UltraFlash improves the 4K speed–quality frontier.** It achieves a 2× speedup over the current state-of-the-art cascade baseline with better FID, moving closer to the Pareto-optimal frontier.

Abstract

Megapixel visual synthesis with latent diffusion models requires operating far beyond the resolution at which these models are trained. Cascade refinement is the most successful approach to handling this problem: it upgrades a base image at a regular resolution to the target resolution via multiple cascade upsampling steps, each involving a resolution upgrade traversed through pixel space and a denoising-based refinement in latent space. Although denoising is often treated as the main cost, we identify an underexplored source of overhead: repeatedly moving representations through the VAE. We refer to these steps as *pixel-space transitions*: intermediate latents are decoded to RGB, resized, and re-encoded before refinement, while final latents are decoded patch by patch with dense overlap to suppress artifacts. We present **UltraFlash**, which shifts the focus of acceleration from denoising alone to the VAE transitions around it. Its first component, **Latent Hyperloop**, replaces the intermediate RGB round trip with a learned latent-space shortcut that approximates the same transformation while avoiding repeated VAE calls for decoding and encoding. Its second component, **Sparse Patch-based Decoding**, makes final reconstruction efficient by tracing patch seams to inconsistent GroupNorm statistics and reusing cached statistics across patches, enabling sparse VAE decoding with much smaller overlap. Integrated into the current state-of-the-art baseline, UltraFlash reduces 4K generation latency by $\sim 2\times$ while improving image quality, demonstrating that transition-level optimization can advance both efficiency and fidelity. More broadly, these transition costs recur across megapixel synthesis pipelines, suggesting that optimizing pixel-space transitions is an effective route to fast, high-fidelity visual synthesis.

[⋆]Equal contribution.
[†]Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Latent diffusion models have greatly advanced visual synthesis, but their native training resolution remains a practical boundary. Although applications increasingly require 2K, 4K, and larger outputs, models trained around 512×512 or 1024×1024 often produce repeated objects, distorted layouts, or unstable textures when directly evaluated on larger canvases. Fine-tuning on high-resolution data is a natural remedy, but it is difficult to scale: diverse megapixel-scale datasets are limited, training at such resolutions is expensive, and the adapted model often becomes specialized to a fixed target resolution [33, 35]. Existing alternatives therefore rely on inference-time strategies that reuse pretrained models. Direct extrapolation methods modify inference on enlarged latents, but can suffer from memory cost and structural instability at extreme resolutions. Cascade refinement methods instead generate at a reliable native resolution and progressively construct larger images through refinement stages [6, 9, 21, 29, 36]. Representative systems differ in how they refine enlarged latents: DemoFusion uses progressive patch-based refinement, FouriScale and FreeScale modify frequency or scale behavior, HiWave uses structure-preserving inversion, and PixelRush [13] accelerates the cascade with one-step diffusion refinement. Among these, PixelRush currently provides the strongest speed–quality trade-off, making it the most relevant stress test for remaining non-denoising overhead. We therefore use PixelRush as the main instantiation of UltraFlash, while targeting VAE-heavy pixel-space transitions that are shared by the broader cascade family.

Most acceleration efforts for megapixel synthesis focus on denoising or refinement. This is natural, since diffusion sampling is expensive and larger spatial sizes increase the cost of each denoising operation. However, we find that this view is incomplete. Modern cascade pipelines also spend substantial time repeatedly moving representations through the VAE. We call these operations *pixel-space transitions*: steps where a latent representation is converted to RGB or where RGB is converted back to a latent representation. These transitions appear in two places. First, intermediate latents are often decoded to RGB, resized, and re-encoded before refinement. Second, the final latent must be decoded to RGB, but at megapixel scale this is typically done patch by patch with dense overlap to suppress artifacts. As a result, a significant fraction of runtime is spent not on denoising itself, but on VAE-heavy transitions around denoising.

This observation exposes an underexplored source of overhead. The two transitions have different roles and therefore require different solutions. The intermediate RGB round trip is avoidable: it is used as a convenient way to transform a latent before refinement, but it need not be performed through pixels. The final transition is unavoidable because the output image must be RGB, but its dense patch implementation is inefficient. Thus, the design goal is to replace the avoidable intermediate VAE round trip and make the final VAE decoding sparse without introducing seams.

We present **UltraFlash**, which shifts the focus of acceleration from denoising alone to the VAE transitions around it. UltraFlashcontains two components. First, **Latent Hyperloop** replaces the intermediate RGB round trip with a learned latent-space shortcut. It approximates the decode–resize–encode transformation while avoiding repeated VAE calls for decoding and encoding. Second, **Sparse Patch-based Decoding** accelerates final reconstruction. We trace sparse patch-decoding artifacts to inconsistent GroupNorm statistics across independently decoded patches. Sparse Patch-based Decoding fixes this by caching normalization statistics from reference patches and reusing them across all patches, enabling sparse VAE decoding with much smaller overlap.

Integrated into the current state-of-the-art cascade baseline [13], UltraFlash reduces 4K generation latency by $2\times$ while improving image quality. This result shows that VAE transitions remain a substantial optimization spot even in highly optimized megapixel synthesis systems. More broadly, RGB round trips and dense patch-wise VAE decoding appear across cascade pipelines, suggesting that optimizing pixel-space transitions is an effective route to faster, high-fidelity visual synthesis.

Our contributions are:

- We identify repeated VAE-heavy pixel-space transitions as an underexplored source of overhead in megapixel visual synthesis, beyond denoising and refinement.
- We introduce **Latent Hyperloop**, a learned latent-space shortcut that replaces the intermediate decode–resize–encode round trip while avoiding repeated VAE decoding and encoding.
- We introduce **Sparse Patch-based Decoding**, which enables sparse VAE decoding by sharing cached normalization statistics across independently decoded patches.

2 Related Work

Megapixel Visual Synthesis. Scaling diffusion models beyond their native training resolution has been explored through several routes. One route trains or adapts models for native high-resolution generation, such as SANA, HunyuanImage, and other dedicated 2K/4K generators [33, 28]. These methods can achieve strong quality, but they require high-resolution data, model adaptation, or dedicated training. A second route directly extrapolates pretrained models to larger canvases through inference-time modifications, such as convolutional dilation [7], frequency or scale fusion [9], flow-aligned guidance [2], or positional extrapolation for FLUX-based models [10]. These methods avoid retraining, but can suffer from memory cost, structural instability, or unnatural textures at extreme resolutions [17]. A third route uses cascade or region-based refinement, synthesizing native-resolution windows and merging them into a larger image through average blending [1, 6], DDIM inversion [29], or few-step refinement [13]. This cascade line has progressed from multi-step patch refinement in DemoFusion to structure-preserving inversion in HiWave, and most recently to PixelRush, which uses one-step Turbo refinement to deliver the strongest speed–quality trade-off among cascade methods.

Our work focuses on the cascade setting because it reuses pretrained models in their reliable operating regime while supporting megapixel outputs. We use PixelRush as the main experimental instantiation because it is already optimized for denoising speed; improving it therefore tests whether VAE transition overhead remains a meaningful bottleneck after the dominant sampling cost has been compressed. Rather than introducing another refinement strategy, UltraFlash optimizes the VAE-heavy transitions surrounding refinement: intermediate RGB round trips that prepare refinement inputs, and final patch-wise VAE decoding that reconstructs the output image.

Pixel-Space Transitions. Latent diffusion models use a VAE to move between compressed latents and RGB images [22]. In megapixel synthesis, this creates two transition costs. First, intermediate latents are often decoded to RGB, resized, and re-encoded before refinement. This route is stable and avoids artifacts from naive latent manipulation [21, 29], but requires an expensive decode–encode round trip. Because the VAE encoder is nonlinear, ordinary latent interpolation is not that round trip. FAM Diffusion [34] compensates with learned frequency and attention modulation over the whole latent, and it changes the denoiser. Latent Hyperloop is a patch-wise shortcut for the same decode–resize–encode transformation and leaves the denoiser unchanged.

Second, the final latent must be decoded to RGB. Since full-image VAE decoding is memory-intensive at megapixel scale, pipelines decode overlapping patches and blend them. Large overlaps hide seams but repeat computation; small overlaps reveal boundary artifacts. We identify inconsistent GroupNorm statistics [32] across patches as a key cause, and use **Sparse Patch-based Decoding** to share cached statistics across patches, enabling sparse reconstruction with much smaller overlap.

Super-Resolution. Super-resolution methods reconstruct high-resolution images from low-resolution inputs [4, 5, 30]. Diffusion-based super-resolution methods [14, 31] further improve perceptual quality, but are typically trained for restoration. Megapixel visual synthesis differs from restoration: the input is a generated intermediate representation, and the goal is coherent visual synthesis rather than recovering a unique high-resolution target [13]. Thus, although super-resolution and megapixel synthesis both involve scale expansion, they address different problem settings and require different evaluation criteria.

3 Background

Latent Diffusion Models [22] perform diffusion in the latent space of a pretrained autoencoder. Given an image $\mathbf{x}$, the encoder $\mathcal{E}$ maps it to a latent $\mathbf{z}_0 = \mathcal{E}(\mathbf{x})$. The forward process gradually corrupts this latent:

$$q(\mathbf{z}_t \mid \mathbf{z}_0) = \mathcal{N}(\mathbf{z}_t; \sqrt{\bar{\alpha}_t}\mathbf{z}_0, (1 - \bar{\alpha}_t)\mathbf{I}), \tag{1}$$

where t is the diffusion timestep and $\bar{\alpha}_t$ is determined by the noise schedule. A denoising network $\epsilon_\theta(\mathbf{z}_t, t)$ is trained to reverse this process. At inference, a sample is generated by denoising a Gaussian latent and decoding the final latent with the VAE decoder $\mathcal{D}$.

Cascade Refinement Pipelines. Megapixel synthesis commonly starts from the model's native resolution and progressively enlarges the image through cascade refinement [6, 21, 29, 13]. Although methods differ in refinement details, each stage typically initializes a larger latent by routing the current latent through RGB.

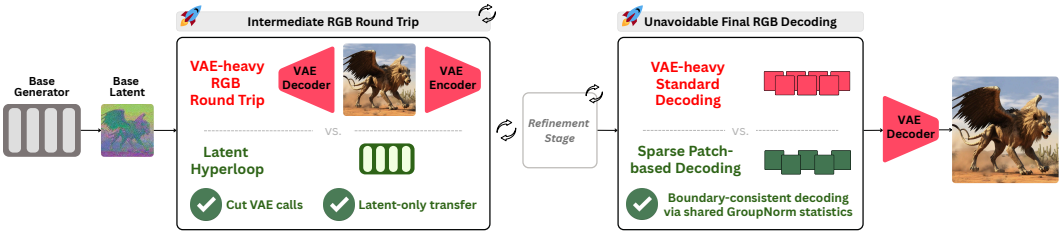


Figure 2: **UltraFlash targets transition costs common to cascade megapixel synthesis.** Cascade pipelines often use pixel space twice: an intermediate RGB round trip before refinement and dense patch-wise VAE decoding for final output. UltraFlash accelerates both without changing the denoising/refinement stage. **Latent Hyperloop** removes repeated VAE decode/encode calls during intermediate transfer, while **Sparse Patch-based Decoding** reduces final decoding redundancy by enforcing consistent GroupNorm statistics across patches.

Let $\mathbf{z}_0^r$ be a clean latent at resolution r, where a $2\times$ increase in height and width maps r to $4r$. The standard pixel-space transition is

$$\bar{\mathbf{z}}_0^{4r} = \mathrm{UP}_{\mathrm{pixel}}(\mathbf{z}_0^r) = \mathcal{E}(\mathrm{Interpolate}(\mathcal{D}(\mathbf{z}_0^r), 2)), \qquad \mathbf{z}_0^{4r} = \phi(\bar{\mathbf{z}}_0^{4r}), \tag{2}$$

where $\mathcal{D}$ and $\mathcal{E}$ are the VAE decoder and encoder, $\mathrm{Interpolate}(\cdot, 2)$ resizes the RGB image by $2\times$, and ϕ denotes refinement. This decode–resize–encode route is stable but VAE-heavy.

The final latent $\mathbf{z}_0^R$ must also be decoded to RGB. Since full-image VAE decoding is memory-intensive at megapixel scale, practical systems decode overlapping patches and stitch them together, where dense overlap suppresses seams but repeats computation. Thus, cascade pipelines contain two major transition costs: repeated intermediate VAE round trips and final patch-wise VAE decoding. UltraFlash targets both: **Latent Hyperloop** replaces the intermediate round trip with a learned latent shortcut, while **Sparse Patch-based Decoding** makes final reconstruction sparse and consistent.

4 Accelerating Megapixel Visual Synthesis

We present **UltraFlash**, which shifts acceleration from denoising alone to the VAE-heavy pixel-space transitions that surround it. In cascade megapixel synthesis, these transitions arise twice: intermediate latents undergo an RGB round trip before refinement, while final RGB decoding must be performed patch by patch due to memory constraints.

UltraFlash handles these two roles differently: **Latent Hyperloop** (Sec. 4.1) replaces the avoidable intermediate RGB round trip, while **Sparse Patch-based Decoding** (Sec. 4.2) accelerates the unavoidable final decoding. An overview is shown in Fig. 2.

4.1 Latent Hyperloop: Bypassing Intermediate RGB Round Trips

Motivation. Many cascade refinement pipelines [6, 21, 29, 13] initialize each refinement stage through a VAE-heavy RGB round trip. Given a latent $\mathbf{z}_0^r$, the pipeline decodes it to RGB, changes its spatial scale, and re-encodes the resized image back into latent space:

$$\mathrm{UP}_{\mathrm{pixel}}(\mathbf{z}_0^r) = \mathcal{E}(\mathrm{Interpolate}(\mathcal{D}(\mathbf{z}_0^r), 2)), \tag{3}$$

where $\mathcal{D}$ and $\mathcal{E}$ denote the VAE decoder and encoder, respectively. The appeal of this RGB round trip is that resizing in pixel space is simple and well-defined. Its drawback is computational cost: each transfer invokes both the VAE decoder and encoder. In our main instantiation with the state-of-the-art cascade method [13], this transition accounts for approximately ***30–40%*** of cascade-stage runtime.

The challenge is that the RGB round trip cannot be replaced by ordinary latent interpolation. Since the VAE encoder is nonlinear, spatial resizing in latent space is not equivalent to resizing in RGB and then encoding the result: $\mathrm{Interpolate}(\mathbf{z}, 2) \neq \mathcal{E}(\mathrm{Interpolate}(\mathcal{D}(\mathbf{z}), 2))$. Thus, we replace generic latent interpolation with a learned latent-space shortcut that approximates the decode–resize–encode transformation while avoiding the VAE decode–encode computation.

Problem Setting. We define **Latent Hyperloop** as a lightweight function $\mathcal{H}_\phi$ that maps the current latent directly to the latent that would otherwise be produced by the RGB round trip:

$$\bar{\mathbf{z}}_0^{4r} = \mathcal{H}_\phi(\mathbf{z}_0^r) \approx \mathrm{UP}_{\mathrm{pixel}}(\mathbf{z}_0^r). \tag{4}$$

Once trained, $\mathcal{H}_\phi$ replaces the decode–interpolate–encode route during intermediate transfer. A good Latent Hyperloop $\mathcal{H}_\phi$ must satisfy the following conditions: (1) ***Accurate*** in replicating the upsampling operator $\mathrm{UP}_{\mathrm{pixel}}$ without going through pixel space, (2) ***Efficient*** in both memory usage and computation, causing minimal latency, and (3) ***General*** to work at every cascade step. These criteria are challenging, requiring novel designs in data flow, network structure, training data, and training objectives. We will discuss below our proposals to handle each mentioned aspect.

Context-Expanded Patching. To make $\mathcal{H}_\phi$ general and work at every cascade step regardless of input resolution, we design it to process patch-by-patch, using fixed resolutions for the input and output latent patches. Since the follow-up refinement step also processes overlapping patches, a convenient design is to use the same patch configuration in these two steps: the output patches of $\mathcal{H}_\phi$ should have the same resolution and location as the ones used in the refinement. With that configuration, the patches produced by $\mathcal{H}_\phi$ can be sent for processing at the refinement step directly, skipping the operation to collect and aggregate them to form the full latent output $\bar{\mathbf{z}}_0^{4r}$.

For each target patch in $\bar{\mathbf{z}}_0^{4r}$, a natural input to $\mathcal{H}_\phi$ is the coordinate-matched source patch in $\mathbf{z}_0^r$. However, this crop alone is insufficient. The decode–resize–encode route approximated by $\mathcal{H}_\phi$ contains convolutional VAE layers, so the target patch depends not only on the aligned source region but also on its surrounding context. We therefore expand each source crop by p context pixels on every side before feeding it to $\mathcal{H}_\phi$, so that a source patch of side length s becomes an input of side length $s + 2p$; we use $s = 64$ and $p = 8$, i.e., 80×80 inputs for 128×128 target patches. This context-expanded design captures the local dependencies of the original RGB round trip while preserving patch-wise processing. Its necessity is confirmed by the ablation on p in Appendix B.2.

Lightweight Spatial Prior. Latent Hyperloop $\mathcal{H}_\phi$ needs to double the resolution of the input latent $\mathbf{z}_0^r$. While we could use any super-resolution backbone with an adjusted number of input/output channels and train on paired data, this would require the model to learn spatial upsampling behavior in the latent space from scratch. Instead, we reuse *the first decoding block* of the VAE decoder, denoted as $\mathcal{D}^{(1)}$, which already implements latent upsampling and has been well trained on massive data. This pretrained block serves as a *spatial prior*, effectively guiding $\mathcal{H}_\phi$'s spatial upsampling operation. However, this block often produces feature maps with a different number of channels and unaligned geometry compared to the desired output $\mathbf{z}_0^{4r}$. Hence, we need *a projection head* to adjust the number of channels and refine the geometry of the produced latent output.

In summary, our proposed **Latent Hyperloop** $\mathcal{H}_\phi$ consists of two components: ***a frozen VAE decoding block*** $\mathcal{D}^{(1)}$ and ***a trainable projection head*** h_ψ. Because context-expanded patching uses an input crop larger than the coordinate-matched source region, the spatial size entering $\mathcal{H}_\phi$ is larger than the nominal source patch size. We therefore adjust the padding in $\mathcal{D}^{(1)}$ and the early layers of h_ψ so that the output matches the required target patch size, requiring no major architectural change.

Training Data. Training the **Latent Hyperloop** requires paired input/output data. Given an image $\mathbf{x}$ at the base resolution r_0, e.g., 1024×1024, and its upsampled version $\mathbf{x}^{up}$, we pass them through the VAE encoder to get the base latent $\mathbf{z}_0^{r_0} = \mathcal{E}(\mathbf{x})$ and its upsampled latent $\bar{\mathbf{z}}_0^{4r_0} = \mathcal{E}(\mathbf{x}^{up})$. We then crop these latents with predefined cropping boxes, denoted as b_i for $\mathbf{z}_0^{r_0}$ and b_i^{up} for $\bar{\mathbf{z}}_0^{4r_0}$ with i being the patch index among predefined patch locations, to form paired input/output latent patches $\left(\mathbf{z}_0^{r_0}(b_i), \bar{\mathbf{z}}_0^{4r_0}(b_i^{up})\right)$. We construct these pairs from a dataset of high-resolution high-quality images.

When the VAE is perfect, i.e., $\mathcal{D}(\mathcal{E}(\mathbf{x})) = \mathbf{x}$, and $\mathbf{x}^{up} = \mathrm{Interpolate}(\mathbf{x}, 2)$, the upsampled latent $\bar{\mathbf{z}}_0^{4r_0}$ is equal to the result of applying VAE round-trip on the base latent $\mathbf{z}_0^{r_0}$, i.e., $\bar{\mathbf{z}}_0^{4r_0} = \mathrm{UP}_{\mathrm{pixel}}(\mathbf{z}_0^{r_0})$. However, in practice, VAE is imperfect, leading to imprecise upsampling when using the VAE round-trip. Our data construction avoids that flaw, allowing UltraFlash to even surpass the original cascade baseline with VAE round trip in terms of quality, as will be shown in Section 5.

Training Objectives. After constructing training data, we define training losses for building a high-quality Latent Hyperloop $\mathcal{H}_\phi$. Given each training example $(\mathbf{z}^{in}, \mathbf{z}^{out})$, two complementary losses are applied to ensure that $\mathcal{H}_\phi$ could faithfully estimate $\mathbf{z}^{out}$ from $\mathbf{z}^{in}$: $\hat{\mathbf{z}}^{out} = \mathcal{H}_\phi(\mathbf{z}^{in}) \approx \mathbf{z}^{out}$.

Latent Reconstruction ($\mathcal{L}_{rec}$). A straightforward objective is latent reconstruction, implemented by an ℓ_1 reconstruction loss:

$$\mathcal{L}_{rec} = \|\mathbf{z}^{out} - \hat{\mathbf{z}}^{out}\|_1. \tag{5}$$

While ℓ_2 is a common choice, it is less effective for latent-space reconstruction, as noted in recent works [16, 37, 19]. The ℓ_1 objective provides better training stability and preserves the structural sparsity of the upsampled latent.

Attention Alignment ($\mathcal{L}_{atten}$). Reconstruction loss alone is insufficient. Since the loss is operated directly on the latent manifold, a small numerical distance error does not necessarily guarantee that two latents are perceptually or semantically equivalent. Furthermore, by treating each latent pixel independently, a simple reconstruction objective often tolerates subtle misalignments. Both lead to a loss of fine-grained structure in the decoded image. In principle, a GAN loss could help, but it often makes optimization substantially more sensitive to hyperparameters and empirically less stable.

Motivated by X2I [18], we instead seek supervision signals from attention responses of the refiner model when processing $\mathbf{z}^{out}$ and $\hat{\mathbf{z}}^{out}$, which are stable and sensitive to fine structure. Self-attention maps in the refiner encode rich interactions across spatial locations in the latent, and thus can *magnify* small deviations into larger, more detectable discrepancies in the network's internal responses. By monitoring the attention misalignment, we can penalize structured mismatches between $\mathbf{z}^{out}$ and $\hat{\mathbf{z}}^{out}$ that $\mathcal{L}_{rec}$ may underweight.

Let $\epsilon_\theta(\cdot, t, c)$ denote the refiner's denoising function at time step t with text conditioning c. To compare its responses under identical diffusion conditions, we apply the forward noising operator using the *same* noise drawn from $\mathcal{N}(0, I)$ to convert $\mathbf{z}^{out}$ and $\hat{\mathbf{z}}^{out}$ into noised latents $\mathbf{z}_t^{out}$ and $\hat{\mathbf{z}}_t^{out}$. We then run one denoising pass with ϵ_θ on these noised inputs and extract the self-attention map $P^l(\cdot; t, c)$ for each self-attention (SA) layer l. We minimize the reverse KL divergence between the attention distributions induced by $\mathbf{z}_t^{out}$ and $\hat{\mathbf{z}}_t^{out}$:

$$\mathcal{L}_{atten} = \sum_{l \in \text{SA layers}} \tau^2 \, \mathrm{KL}\big(P^l(\mathbf{z}_t^{out}; t, c) \,\big\|\, P^l(\hat{\mathbf{z}}_t^{out}; t, c)\big)\,, \tag{6}$$

where τ is a temperature used for numerical stability. Empirically, we found this attention alignment exceptionally effective, speeding up training convergence and enhancing fine-detail reconstruction.

Final Objective ($\mathcal{L}_{final}$). Our training minimizes a weighted sum of latent reconstruction and attention alignment, using hyperparameters $\alpha, \beta \geq 0$:

$$\mathcal{L}_{final} = \alpha \mathcal{L}_{rec} + \beta \mathcal{L}_{atten}. \tag{7}$$

For simplicity, we set $\alpha = \beta = 1$ in our experiments.

4.2 Sparse Patch-based Decoding: Accelerating Final RGB Decoding

While Latent Hyperloop removes avoidable intermediate RGB round trips, the final pixel-space transition is unavoidable: after refinement, the latent $\mathbf{z}_0^R$ must be decoded into RGB. At megapixel scale, however, full-latent VAE decoding is often memory-prohibitive, so practical synthesis pipelines decode the latent patch by patch and blend overlapping regions [1, 6, 29, 13]. Dense overlap helps suppress seams, but it also repeats VAE computation over many regions. A natural acceleration is to reduce the overlap, but naive sparse decoding introduces visible boundary artifacts, including color shifts, brightness inconsistencies, and texture discontinuities at patch boundaries, as shown in Fig. 3.

Why Sparse Decoding Fails. We trace the root cause of sparse-decoding artifacts to the *GroupNorm (GN) layers* in the VAE decoder. GroupNorm [32] normalizes activations within each group of channels using statistics computed from the current input.

When the full image is decoded in a single pass, the mean μ and variance σ^2 are computed over the *entire* spatial extent, producing globally consistent normalization. However, when decoding individual patches, each patch computes its own $\mu^{(j)}$ and $\sigma^{2(j)}$ based only on its local spatial content.

Patches with different spatial content (*e.g.*, a bright sky vs. a dark ground) yield substantially different statistics. This causes the same latent values in overlapping regions to be decoded to different pixel values depending on which patch they belong to, producing visible seams even after blending.

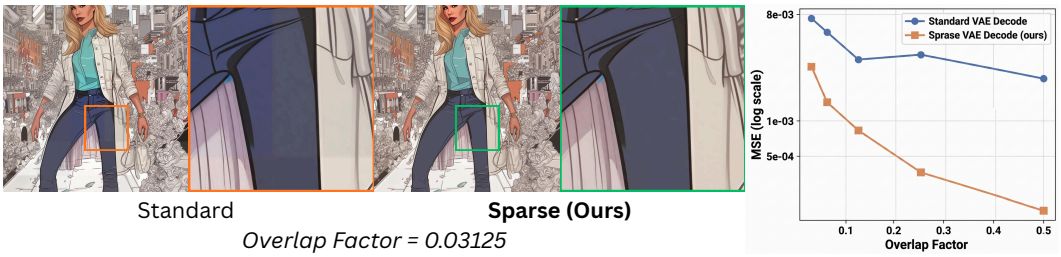


Figure 3: **Shared GroupNorm statistics enable sparse final RGB decoding.** When overlap is small, standard patch-wise decoding shows seam artifacts caused by independently computed GroupNorm statistics. **Sparse Patch-based Decoding** uses shared cached statistics to align normalization across patches, making adjacent patches decode consistently. The right plot shows that our method maintains consistently low boundary MSE across overlap factors when decoding with sparse overlaps.

Shared GN Statistics. Our solution is simple and requires no training. We compute a set of *reference GroupNorm statistics* once, then reuse them for all patches during decoding. First, we decode a set of reference patches $\{P_{\text{ref}}^{(i)}\}_{i=1}^{N_{\text{ref}}}$ (utilizing the four corner patches of the latent) through the VAE decoder, recording the GroupNorm statistics $(\mu_l^{\text{ref}}, \sigma_l^{\text{ref}})$ at every layer l. The cached statistics are $\mathcal{C} = \{(\mu_l^{\text{ref}}, \sigma_l^{\text{ref}})\}_{l=1}^{L}$, where L is the number of GroupNorm layers in the VAE decoder. Second, when decoding each patch $P^{(j)}$, we replace the per-patch GroupNorm computation with the cached reference statistics. This ensures that identical latent values always produce identical pixel values, regardless of which patch they belong to. The overlapping regions now agree perfectly, and simple blending, or even direct stitching, produces seamless results.

Enabling Smaller Overlap. A key benefit of Sparse Patch-based Decoding is that it enables the use of much smaller overlaps during patch-wise VAE decoding. In standard cascade and patch-based synthesis pipelines [1, 6, 29, 13], large overlaps (e.g., 25%) are often needed to mask patch-boundary inconsistencies through blending. With cached GroupNorm statistics, the overlap can be reduced by up to **8**$\times$ in our main setting while preserving visual continuity, since it no longer needs to hide per-patch normalization mismatch. This directly translates to fewer total patches and faster decoding:

$$N_{\text{patches}} = \left\lceil \frac{H - o}{H_p - o} \right\rceil \times \left\lceil \frac{W - o}{W_p - o} \right\rceil, \tag{8}$$

where $H_p \times W_p$ is the patch size and o is the overlap. Smaller overlap increases the patch stride and reduces N_{patches}; in our main 4K setting, this reduces the number of decoded patches by up to **30–40%**, leading to a proportional reduction in VAE decoding time. Fig. 3 quantifies boundary consistency across overlap settings. We vary the overlap factor and measure the MSE between decoded pixel values in overlapping regions of adjacent patches, averaged over 100 held-out latents. Standard decoding exhibits high boundary error that decreases only gradually with larger overlaps, while Sparse Patch-based Decoding maintains low boundary error by enforcing globally consistent GroupNorm statistics.

5 Experiments

5.1 Experimental Settings

Latent Hyperloop. Latent Hyperloop is a lightweight network with $\sim$ 5M trainable parameters. Unless otherwise stated, we use SwinIR [15] as the projection head backbone and train it with the objectives described in Sec. 4.1. For the attention-alignment loss, we use a fixed noising level $t = 1$ and temperature $\tau = 3.0$. Training data consists of 1.2M images synthesized with SDXL [20] using prompts from JourneyDB [27] and LAION [26], each paired with its $2\times$-interpolated version as described in Sec. 4.1. Training uses 8 NVIDIA H100 GPUs (80GB each) with a batch size of 32 per GPU, i.e., a global batch size of 256; optimization details are given in Appendix A.3.

Evaluation Data and Metrics. Following prior protocols [6, 9, 21], we evaluate on 1000 prompts sampled from LAION-Aesthetics [26, 25] and report Fréchet Inception Distance (FID) [8], crop-based FID (FID$_c$), Inception Score (IS) [23], and wall-clock inference time. FID measures global

Table 1: **Quantitative comparison among SDXL-based high-resolution generation methods.** We report FID (↓), FID_c (↓), Inception Score (IS ↑), and Inference Time (seconds).

	Resolution 2K (2048×2048)				**Resolution 4K** (4096×4096)			
Method	**FID** ↓	**FID_c** ↓	**IS** ↑	**Time (s)**	**FID** ↓	**FID_c** ↓	**IS** ↑	**Time (s)**
FouriScale [9]	72.65	51.90	12.31	87.84	98.97	69.56	8.54	680.57
DemoFusion [6]	68.46	43.39	13.15	75.89	74.75	52.14	12.57	507.74
FreeScale [21]	52.87	29.64	13.56	53.20	58.28	34.01	13.35	323.93
PixelRush-SD [13]	52.56	29.66	13.69	<u>3.17</u>	56.50	33.76	13.42	16.37
PixelRush-XL [13]	50.13	**29.13**	**14.32**	4.06	54.28	33.15	13.71	17.11
UltraFlash-SD (Ours)	<u>50.02</u>	<u>29.21</u>	14.03	**2.78**	<u>52.38</u>	**32.47**	**13.99**	**8.04**
UltraFlash-XL (Ours)	**49.86**	29.73	<u>14.14</u>	3.50	**51.32**	<u>33.01</u>	<u>13.93</u>	<u>9.61</u>

distribution quality, while FID_c evaluates local high-frequency fidelity using crops without resizing. All latency numbers are measured on a single NVIDIA A100-40GB GPU unless stated otherwise.

Baselines. Our main comparison focuses on SDXL-based cascade methods, including FouriScale [9], DemoFusion [6], FreeScale [21], and the fast cascade baseline PixelRush [13]. This setting provides a direct comparison because these methods share a similar base-model regime and target high-resolution extrapolation from pretrained native-resolution generators. We use PixelRush as the representative fast cascade pipeline because it offers the strongest speed–quality trade-off among prior cascade methods. Unlike FouriScale, DemoFusion, and FreeScale, which rely on multi-step SDXL refinement, PixelRush uses one-step Turbo refinement for rapid generation. We denote configurations by the refiner used in the cascade: **-SD** uses SD-Turbo, while **-XL** uses SDXL-Turbo [24]. Unless otherwise specified, both configurations follow the same base-generation protocol. Finally, UltraFlash denotes the PixelRush fast cascade equipped with our Latent Hyperloop and Sparse Patch-based Decoding.

Refinement-stage sparse patching. PixelRush already uses sparse patching during diffusion refinement to reduce overlap between refinement patches, discussed in its supplemental materials. We regard this as part of the baseline cascade pipeline. It is separate from **Sparse Patch-based Decoding**: the former reduces denoising/refinement cost, while the latter reduces final VAE-to-RGB decoding cost by enabling small-overlap patch decoding without boundary artifacts. For fair comparison, we report PixelRush under the same refinement-stage sparse patching setting.

5.2 Main Results on Fast Cascade Inference

Table 1 shows that adding our transition operators to the fast cascade improves both efficiency and fidelity. At 2K, the transition overhead is relatively small because the cascade contains fewer high-resolution patches, yet the gains remain consistent: UltraFlash-SD reduces runtime from 3.17s to 2.78s while improving FID from 52.56 to 50.02, and UltraFlash-XL achieves the best 2K FID of 49.86. The benefit becomes more pronounced at 4K, where transition costs compound across cascade stages and final VAE decoding. UltraFlash-SD reduces latency from 16.37s to 8.04s relative to the corresponding PixelRush-SD setting, while improving FID from 56.50 to 52.38. The XL configuration shows the same trend, reducing latency from 17.11s to 9.61s and improving FID from 54.28 to 51.32. These results indicate that replacing the intermediate RGB round trip with **Latent Hyperloop** and accelerating final RGB decoding with **Sparse Patch-based Decoding** removes transition overhead without sacrificing visual quality. Detailed ablations of both components are provided in Appendix B.

5.3 Qualitative Comparison

As illustrated in Fig. 4, UltraFlash demonstrates superior structural coherence and realism compared to existing state-of-the-art methods. While baselines like FouriScale and DemoFusion exhibit structural deterioration at high resolutions (e.g., object repetition or unnatural texture), both PixelRush and UltraFlash maintain robust global layout. However, the advantage of UltraFlash becomes evident in the fine-grained details. As shown in the zoomed crops, our pipeline resolves sharper high-frequency textures, effectively mitigating the subtle smoothing artifacts observed in PixelRush.

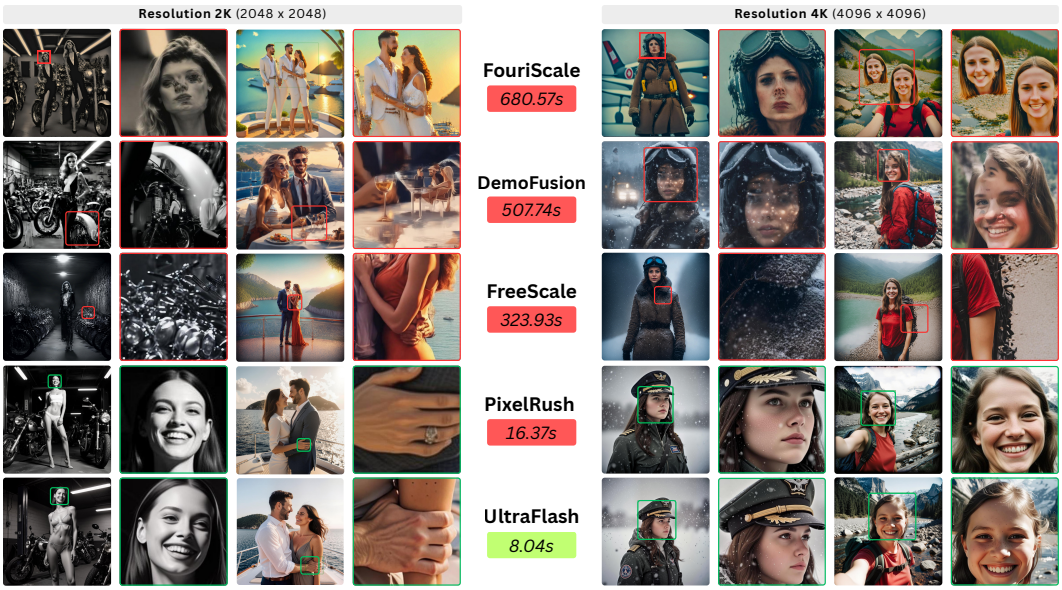


Figure 4: **Qualitative comparison at 2K and 4K resolution.** We compare high-resolution generation from FouriScale, DemoFusion, FreeScale, PixelRush, and UltraFlash at 2048×2048 (left) and 4096×4096 (right). UltraFlash produces sharper textures, cleaner facial regions, and more coherent fine structures, while achieving substantially faster inference. Runtimes are measured at 4096×4096.

Table 2: **Additional evaluations beyond the main SDXL cascade comparison.** Left: contextual comparison with alternative 4K generation routes, separated from Table 1 because these methods use different backbones or training assumptions. Right: transfer of the proposed transition operators to a second high-resolution pipeline.

Method	Backbone	Route	FID↓	Time (s)↓
SANA-4K [33]	SANA	Native 4K	72.13	24.79
DyPE-FLUX [10]	FLUX	Direct 4K	59.05	587.13
UltraFlash-SD*	SDXL/SD-Turbo	Cascade	52.38	8.04
UltraFlash-XL*	SDXL/SDXL-Turbo	Cascade	51.32	9.61

(a) Alternative 4K generation routes

Configuration	FID↓	FID_c ↓	IS↑	Time (s)↓
FreeScale [21]	58.28	34.01	13.35	323.93
+ UltraFlash	*54.42*	*34.58*	*13.66*	*313.40*

(b) Cross-pipeline generalization

5.4 Comparison with Non-Cascade 4K Routes

To position our cascade pipeline in the broader 4K generation landscape, Table 2(a) compares against two representative non-cascade routes. SANA-4K [33] directly targets native 4K generation with a compact high-resolution model design, so it serves as a speed-oriented native-generation baseline. DyPE-FLUX [10] extrapolates a strong FLUX [12] backbone to 4K and serves as a quality-oriented direct-inference baseline. Because these methods differ in backbone, training data, and inference assumptions, we treat this as a contextual comparison. UltraFlash-SD outperforms both methods in FID and particularly in latency, suggesting that a fast cascade pipeline with optimized pixel-space transitions can remain highly competitive with native or direct 4K approaches. We exclude super-resolution post-processing baselines from this comparison since they address a different restoration-style setting and have been shown to underperform cascade generation [21].

5.5 Plug-in Transfer

To test whether UltraFlash is more than a PixelRush-specific optimization, we apply the same transition operators to FreeScale [21]. As shown in Table 2(b), this improves FID from 58.28 to 54.42 and IS from 13.35 to 13.66. Runtime changes modestly from 323.93s to 313.40s because FreeScale remains dominated by multi-step denoising. This confirms that the operators are reusable, while the PixelRush results show their full speed benefit when VAE transitions become the main remaining bottleneck.

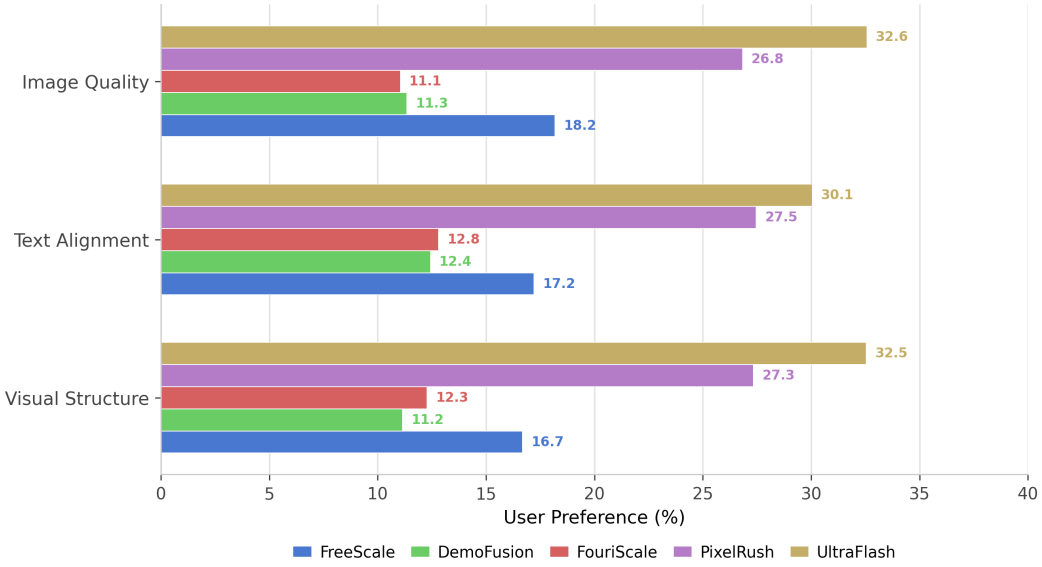


Figure 5: **User preference study.** Preference share of each method on three criteria. UltraFlash (gold bars) receives the highest share (∼30–33%) on all three.

5.6 User Study

To complement the automated metrics with human judgment, we conducted a user study with 137 participants on 25 diverse prompts (3,425 comparisons). Participants compared UltraFlash with the baselines on three criteria: **Text Alignment**, **Image Quality**, and **Visual Structure**. As shown in Fig. 5, UltraFlash receives the highest preference rate on all three criteria. This is consistent with the qualitative comparison in Fig. 4, where UltraFlash renders sharp, high-frequency textures without the tiling artifacts and structural inconsistencies observed in competing methods.

6 Conclusion

We presented UltraFlash, a transition-efficient method for megapixel visual synthesis. Instead of only accelerating denoising, UltraFlash targets VAE-heavy pixel-space transitions in cascade pipelines. **Latent Hyperloop** replaces the intermediate decode–resize–encode round trip with a learned latent shortcut, while **Sparse Patch-based Decoding** accelerates final RGB reconstruction by sharing cached GroupNorm statistics across patches. Integrated into a state-of-the-art cascade baseline, UltraFlash reduces 4K latency by 2× while improving image quality, showing that pixel-space transitions are an effective target for improving the speed–quality trade-off of megapixel synthesis.

Acknowledgments and Disclosure of Funding

This work was funded by Qualcomm Technologies, Inc. The authors declare no competing interests.

References

[1] O. Bar-Tal, L. Yariv, Y. Lipman, and T. Dekel. Multidiffusion: fusing diffusion paths for controlled image generation. In *Proceedings of the 40th International Conference on Machine Learning*, pages 1737–1752, 2023.

[2] J. Bu, P. Ling, Y. Zhou, P. Zhang, T. Wu, X. Dong, Y. Zang, Y. Cao, D. Lin, and J. Wang. Hiflow: Training-free high-resolution image generation with flow-aligned guidance. *arXiv preprint arXiv:2504.06232*, 2025.

[3] J. Chen, C. Ge, E. Xie, Y. Wu, L. Yao, X. Ren, Z. Wang, P. Luo, H. Lu, and Z. Li. Pixart-σ: Weak-to-strong training of diffusion transformer for 4k text-to-image generation, 2024.

[4] Z. Chen, Z. Wu, E. Zamfir, K. Zhang, Y. Zhang, R. Timofte, X. Yang, H. Yu, C. Wan, Y. Hong, et al. Ntire 2024 challenge on image super-resolution (x4): Methods and results. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6108–6132, 2024.

[5] C. Dong, C. C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In *European conference on computer vision*, pages 184–199. Springer, 2014.

[6] R. Du, D. Chang, T. Hospedales, Y.-Z. Song, and Z. Ma. Demofusion: Democratising high-resolution image generation with no $$$. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6159–6168, 2024.

[7] Y. He, S. Yang, H. Chen, X. Cun, M. Xia, Y. Zhang, X. Wang, R. He, Q. Chen, and Y. Shan. Scalecrafter: Tuning-free higher-resolution visual generation with diffusion models. In *The Twelfth International Conference on Learning Representations*, 2023.

[8] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[9] L. Huang, R. Fang, A. Zhang, G. Song, S. Liu, Y. Liu, and H. Li. Fouriscale: A frequency perspective on training-free high-resolution image synthesis. In *European Conference on Computer Vision*, pages 196–212. Springer, 2024.

[10] N. Issachar, G. Yariv, S. Benaim, Y. Adi, D. Lischinski, and R. Fattal. Dype: Dynamic position extrapolation for ultra high resolution diffusion. *arXiv preprint arXiv:2510.20766*, 2025.

[11] M. Kang, R. Zhang, C. Barnes, S. Paris, S. Kwak, J. Park, E. Shechtman, J.-Y. Zhu, and T. Park. Distilling diffusion models into conditional GANs. In *European Conference on Computer Vision (ECCV)*, 2024.

[12] B. F. Labs. Flux. `https://github.com/black-forest-labs/flux`, 2024.

[13] H.-P. Lai, P. Nguyen, and A. Tran. Pixelrush: Ultra-fast, training-free high-resolution image generation via one-step diffusion. *arXiv preprint arXiv:2602.12769*, 2026.

[14] H. Li, Y. Yang, M. Chang, S. Chen, H. Feng, Z. Xu, Q. Li, and Y. Chen. Srdiff: Single image super-resolution with diffusion probabilistic models. *Neurocomputing*, 479:47–59, 2022.

[15] J. Liang, J. Cao, G. Sun, K. Zhang, L. Van Gool, and R. Timofte. Swinir: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1833–1844, 2021.

[16] S. Lin, A. Wang, and X. Yang. Sdxl-lightning: Progressive adversarial diffusion distillation. *arXiv preprint arXiv:2402.13929*, 2024.

[17] Z. Lin, M. Lin, M. Zhao, and R. Ji. Accdiffusion: An accurate method for higher-resolution image generation. In *European Conference on Computer Vision*, pages 38–53. Springer, 2024.

[18] J. Ma, Q. Peng, X. Guo, C. Chen, H. Lu, and Z. Yang. X2i: Seamless integration of multimodal understanding into diffusion transformer via attention distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16733–16744, 2025.

[19] A. Nguyen, V. Nguyen, D. Vu, T. Dao, C. Tran, T. Tran, and A. Tran. Improved training technique for shortcut models. *arXiv preprint arXiv:2510.21250*, 2025. Accepted at NeurIPS 2025.

[20] D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna, and R. Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

[21] H. Qiu, S. Zhang, Y. Wei, R. Chu, H. Yuan, X. Wang, Y. Zhang, and Z. Liu. Freescale: Unleashing the resolution of diffusion models via tuning-free scale fusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16893–16903, 2025.

[22] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[23] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

[24] A. Sauer, D. Lorenz, A. Blattmann, and R. Rombach. Adversarial diffusion distillation. In *European Conference on Computer Vision*, pages 87–103. Springer, 2024.

[25] C. Schuhmann. LAION-aesthetics. `https://laion.ai/blog/laion-aesthetics/`, 2022.

[26] C. Schuhmann, R. Beaumont, R. Vencu, C. Gordon, R. Wightman, M. Cherti, T. Coombes, A. Katta, C. Mullis, M. Wortsman, et al. LAION-5b: An open large-scale dataset for training next generation image-text models. In *Advances in Neural Information Processing Systems*, volume 35, pages 25278–25294, 2022.

[27] K. Sun, J. Pan, Y. Ge, H. Li, H. Duan, X. Wu, R. Zhang, A. Zhou, Z. Qin, Y. Wang, et al. Journeydb: A benchmark for generative image understanding. *Advances in Neural Information Processing Systems*, 36, 2024.

[28] T. H. Team. Hunyuanimage 2.1: An efficient diffusion model for high-resolution (2k) text-to-image generation. `https://github.com/Tencent-Hunyuan/HunyuanImage-2.1`, 2025.

[29] T. Vontobel, S. Sadat, F. Salehi, and R. M. Weber. Hiwave: Training-free high-resolution image generation via wavelet-based diffusion sampling. 2025.

[30] Z. Wang, J. Chen, and S. C. Hoi. Deep learning for image super-resolution: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3365–3387, 2020.

[31] R. Wu, L. Sun, Z. Ma, and L. Zhang. One-step effective diffusion network for real-world image super-resolution. *Advances in Neural Information Processing Systems*, 37:92529–92553, 2024.

[32] Y. Wu and K. He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[33] E. Xie, J. Chen, J. Chen, H. Cai, H. Tang, Y. Lin, Z. Zhang, M. Li, L. Zhu, Y. Lu, et al. Sana: Efficient high-resolution text-to-image synthesis with linear diffusion transformers. In *The Thirteenth International Conference on Learning Representations*, 2025.

[34] H. Yang, A. Bulat, I. Hadji, H. X. Pham, X. Zhu, G. Tzimiropoulos, and B. Martinez. Fam diffusion: Frequency and attention modulation for high-resolution image generation with stable diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2459–2468, 2025.

[35] J. Zhang, Q. Huang, J. Liu, X. Guo, and D. Huang. Diffusion-4k: Ultra-high-resolution image synthesis with latent diffusion models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 23464–23473, 2025.

[36] J. Zhang, Q. Huang, J. Liu, X. Guo, and D. Huang. Diffusion-4k: Ultra-high-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

[37] Y. Zhang and B. Hooi. Hipa: Enabling one-step text-to-image diffusion models via high-frequency-promoting adaptation. *arXiv preprint arXiv:2311.18158*, 2023.

Supplementary Material

A UltraFlash Pipeline

A.1 Implementation Details

Base Generation. Following previous works, we use SDXL as the default base model to generate images at its native 1024×1024 resolution with a classifier-free guidance scale of 7.5. The reverse diffusion process uses deterministic DDIM sampling ($\eta = 0$).

Cascade Upsampling. Our pipeline operates in a cascaded manner, doubling the spatial resolution at each level. For example, generating an 8192×8192 image proceeds as $1024 \rightarrow 2048 \rightarrow 4096 \rightarrow 8192$, with refinement applied at each level. At each cascade level, the Latent Hyperloop adapter upsamples the latent by $2\times$. Since the adapter backbone is lightweight ($\sim$4.8M parameters), the computational overhead of processing overlapping regions is negligible, so we use a generous 50% overlap for the upsampling stage to ensure smooth transitions.

Patch Extraction. For the refinement stage, we support two patching strategies. In *dense patching*, the upsampled latent is tiled into patches with 50% overlap along both spatial dimensions. In *sparse patching*, we reduce the total number of patches for faster inference. Given the target number of patches per row (or column) as e, the fixed patch window size w, and the upsampling factor relative to the original patch window size u, we need to compute the overlap size for interior patches o and the last patch o_{last} that satisfy:

$$o \cdot (e-2) + o_{\text{last}} = (e-u) \cdot w \tag{A1}$$

To distribute the overlaps almost evenly, we compute them as follows:

$$o = \lfloor \frac{(e-u) \cdot w}{e-1} \rfloor, \tag{A2}$$

$$o_{\text{last}} = (e-u) \cdot w - o \cdot (e-2) \tag{A3}$$

Table A1 summarizes the patching configurations. Note that the patch size for SD-Turbo (64×64) is only half of the patch size for SDXL-Turbo (128×128); hence, for the same target image resolution, SD-Turbo requires a doubled upsampling scale u. At $4\times$ the window size (e.g., SDXL-Turbo at 4K or SD-Turbo at 2K), sparse patching uses 6 patches per row/column instead of the 7 required by dense 50% overlap. At $8\times$ the window size (e.g., SD-Turbo at 4K), the reduction is more substantial: 11 patches per row/column versus 15 with dense patching. Note that SDXL-Turbo at 2K cannot benefit from sparse patching, as dense patching already produces only 3 patches along each dimension, which is the minimum needed for full coverage. Combined with sparse patching, SD-Turbo configurations achieve the best speed-quality trade-off in our experiments.

Table A1: **Patching configurations.** We report here the patch count per spatial dimension (total patches = count^2) in each setting.

Setting	Scale	Dense	Sparse
SDXL-Turbo, 2K	$\times 4$	3	3
SDXL-Turbo, 4K / SD-Turbo, 2K	$\times 4$	7	6
SD-Turbo, 4K	$\times 8$	15	11

A.2 Pseudo Algorithm

Integrating both components, the complete UltraFlash pipeline is summarized in Algorithm A1. Note that the VAE is invoked only **once** at the very end (for final decoding), compared to $2 \times s$ times in PixelRush (encode + decode at each of s cascade levels). The Sparse VAE decoding further accelerates this final decode by enabling small overlaps.

Algorithm A1 UltraFlash Pipeline

Require: Base latent $\mathbf{z}_0^n$ at resolution n, target $R = n \times 4^s$
Require: Inversion timestep K, noise injection weight λ
Require: LH adapter $\mathcal{H}_\theta$, few-step denoiser $\epsilon_\gamma^{\text{refine}}$
Require: Sparse Patch-based VAE decoder $\mathcal{D}_{\text{sparse}}$
Ensure: High-resolution image $\mathbf{x}_{\text{HR}}$

// Cascade upsampling
$r \leftarrow n$
for $i = 1$ to s **do**
  // Latent-space upsampling (replaces VAE round-trip)
  $\bar{\mathbf{z}}_0^{4r} \leftarrow \mathcal{H}_\theta(\mathbf{z}_0^r)$ ▷ Latent Hyperloop
  // Patchify with overlap
  $\{P_j\}_{j=1}^{N_p} \leftarrow \text{Patchify}(\bar{\mathbf{z}}_0^{4r})$
  // Per-patch refinement
  for $j = 1$ to N_p **do**
    $\bar{\mathbf{z}}_K^{(j)} \leftarrow \text{DDIM-Inv}(P_j, K, \epsilon_\gamma^{\text{refine}})$ ▷ Partial inversion
    $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\boldsymbol{\epsilon}^{\text{pred}} \leftarrow \text{Predict Noise}(\mathbf{z}_K^{(j)}, K, \epsilon_\gamma^{\text{refine}})$ ▷ One-step refinement
    $\boldsymbol{\epsilon}^{\text{inj}} \leftarrow \text{Slerp}(\boldsymbol{\epsilon}^{\text{pred}}, \boldsymbol{\epsilon}, \lambda)$ ▷ Noise injection
    $\hat{\mathbf{z}}_0^{(j)} \leftarrow \text{Denoise}(\mathbf{z}_K^{(j)}, K, \boldsymbol{\epsilon}^{\text{inj}})$
  end for
  // Patch recombination
  $\mathbf{z}_0^{4r} \leftarrow \text{GaussianBlend}(\{\hat{\mathbf{z}}_0^{(j)}\}_{j=1}^{N_p})$ ▷ Smooth blending
  $r \leftarrow 4r$
end for

// Stage 3: Final decode with Sparse VAE
$\mathbf{x}_{\text{HR}} \leftarrow \mathcal{D}_{\text{sparse}}(\mathbf{z}_0^R)$ ▷ Sparse VAE decoding
return $\mathbf{x}_{\text{HR}}$

A.3 Latent Hyperloop

We adopt a SwinIR-based architecture for the Latent Hyperloop (LH) and incorporate structural priors from the VAE decoder to guide the learned upsampling. With SDXL-Turbo refiner, LH accepts latent patch inputs with resolution 80×80, i.e., a 64×64 source crop expanded by $p = 8$ context pixels on each side (input side length $s + 2p$ with $s = 64$), to generate upsampled latent patches with resolution 128×128. With SD-Turbo, while we could use smaller input and output patch sizes to align with SD-Turbo's convention, we find it unnecessary and use the same config as with SDXL-Turbo (80×80 for input and 128×128 for output) for simple implementation. The model is trained from scratch using the AdamW optimizer with a fixed learning rate of 10^{-4} (no scheduling) for 20K steps on 8 NVIDIA H100 GPUs (80GB each), with a batch size of 32 per GPU (global batch size 256).

A.4 Sparse VAE Decoding

Overview. When decoding a high-resolution latent (spatial dimensions ≥ 512), a single-pass VAE decode causes out-of-memory errors. The standard solution is patch-wise decoding with overlapping regions. Large overlaps produce seamless results but are slow. Our Sparse VAE Decoding enables small overlaps by ensuring consistent GroupNorm statistics across all patches.

CrossPatchGroupNorm. We replace every `GroupNorm` layer in the VAE decoder with a custom `CrossPatchGroupNorm` module that can operate in two modes: (1) standard mode, computing statistics from the current input, and (2) cached mode, using pre-computed reference statistics.

Reference statistics from the first batch. Rather than requiring a separate reference pass, we compute and cache GroupNorm statistics from the *first batch* of patches. During the first batch's forward pass, a forward hook on each `CrossPatchGroupNorm` layer records the mean and variance aggregated over both the batch and spatial dimensions:

$$\mu_l^{\text{ref}} = \frac{1}{B \cdot |\mathcal{S}|} \sum_{b,s} \mathbf{h}_l^{(b,s)}, \quad \sigma_l^{2,\text{ref}} = \frac{1}{B \cdot |\mathcal{S}|} \sum_{b,s} (\mathbf{h}_l^{(b,s)} - \mu_l^{\text{ref}})^2, \tag{A4}$$

where B is the batch size and $\mathcal{S}$ denotes the spatial and channel-group dimensions. After the first batch completes, the hooks are removed and all subsequent batches use the cached statistics automatically.

Batched decoding. Patches of standard size are grouped into mini-batches and decoded in parallel. Edge patches with non-standard sizes (at image boundaries) are decoded individually with cached statistics. After all patches are decoded, adjacent patches are blended along both vertical and horizontal directions to produce the final seamless output.

Listing 1: Sparse VAE Decoding (simplified)

```python
class CrossPatchGroupNorm(nn.Module):
    """GroupNorm with optional cached stats."""
    def __init__(self, num_groups, num_channels,
                 eps=1e-5, affine=True):
        super().__init__()
        self.num_groups = num_groups
        self.num_channels = num_channels
        self.eps = eps
        self.weight = nn.Parameter(torch.ones(num_channels))
        self.bias = nn.Parameter(torch.zeros(num_channels))

        # Cached stats from reference patch
        self._ref_mean = None
        self._ref_var = None
        self._use_cached = False

    def compute_and_cache_stats(self, x):
        """
        Call this on your reference patch
        BEFORE processing other patches.
        """
        B, C, H, W = x.shape
        G = self.num_groups
        xg = x.view(B, G, C // G, H, W)
        # Aggregate over batch + spatial dims
        self._ref_mean = xg.mean(
            dim=[0, 2, 3, 4], keepdim=True)
        self._ref_var = xg.var(
            dim=[0, 2, 3, 4], keepdim=True,
            unbiased=False)
        self._use_cached = True

    def forward(self, x):
        B, C, H, W = x.shape
        G = self.num_groups
        xg = x.view(B, G, C // G, H, W)
        if self._use_cached and self._ref_mean is not None:
            mean, var = self._ref_mean, self._ref_var
        else:
            # Normal GroupNorm behavior (for reference patch itself)
            mean = xg.mean(dim=[2,3,4], keepdim=True)
            var = xg.var(dim=[2,3,4], keepdim=True,
                         unbiased=False)
        xn = (xg - mean) / (var + self.eps).sqrt()
        xn = xn.view(B, C, H, W)
        return xn * self.weight.view(1,C,1,1) \
             + self.bias.view(1,C,1,1)
```

```python

def replace_groupnorm(model):
    """Replace GN layers"""
    for name, mod in model.named_children():
        if isinstance(mod, nn.GroupNorm):
            new = CrossPatchGroupNorm(
                mod.num_groups, mod.num_channels,
                mod.eps, mod.affine)
            new.weight.data = mod.weight.data.clone()
            new.bias.data = mod.bias.data.clone()
            setattr(model, name, new)
        else:
            replace_groupnorm(mod)
```

Decoding procedure. The full Sparse VAE Decoding proceeds as follows:

1. Replace all GroupNorm layers in the VAE decoder with `CrossPatchGroupNorm`.
2. Tile the latent into overlapping patches; group standard-size patches into batches.
3. Decode the *first batch* with forward hooks that call `compute_and_cache_stats` on each layer's input, then remove hooks.
4. Decode remaining batches and edge patches; cached statistics are used automatically.
5. Blend all decoded patches along vertical and horizontal overlap regions.

This approach avoids a separate reference pass entirely: the first batch of patches serves as the reference, and all subsequent patches benefit from consistent normalization with zero additional overhead.

B Additional Quantitative Results

B.1 Hybrid Noise Inversion

InvSR[3] is a pretrained model for diffusion inversion, targeting the super-resolution task. It takes a coarsely upsampled latent as input and predicts its noise residual, which is added to the input latent to form the noisy latent to produce the super-resolution outcome via one diffusion forward step. This formulation is naturally compatible with our refinement stage: we can introduce a parallel InvSR branch alongside the existing DDIM inversion branch, blending their outputs to combine the structural fidelity of DDIM inversion with the detail-aware noise prediction of InvSR.

Specifically, we compute both the DDIM-inverted noise $\bar{\mathbf{z}}_K^{\mathrm{DDIM}}$ and the InvSR-predicted noise $\bar{\mathbf{z}}_K^{\mathrm{InvSR}}$, then combine them via spherical linear interpolation (Slerp):

$$\mathbf{z}_K = \mathrm{Slerp}(\bar{\mathbf{z}}_K^{\mathrm{DDIM}},\ \bar{\mathbf{z}}_K^{\mathrm{InvSR}},\ \alpha), \quad \alpha = 0.8, \tag{A5}$$

where $\alpha = 0.8$ assigns the dominant weight to DDIM inversion. We use Slerp rather than linear interpolation because noisy latents in diffusion models lie on a high-dimensional hypersphere, and Slerp preserves the norm and angular distribution of the interpolated noise.

Table A2 shows that InvSR alone does not provide strong scores when generating high-resolution images. However, it is highly compatible with our pipeline, and incorporating InvSR into UltraFlash consistently improves both FID and IS across resolutions. At 2K, FID improves from 50.02 to 49.33 with a corresponding IS increase from 14.03 to 14.24. At 4K, FID drops from 52.38 to 51.34. These gains come at the cost of one additional InvSR forward pass per patch, which adds moderate overhead but can be worthwhile when quality is prioritized over speed.

B.2 Ablation Study

Why Each Component Matters. Fig. A1a reveals why both components are necessary. First, we observe that a naive implementation of the Latent Hyperloop (marked with *) successfully reduces

[3] Z. Yue, K. Liao, and C. C. Loy, "Arbitrary-steps image super-resolution via diffusion inversion," in *CVPR*, pp. 23153–23163, 2025.

Table A2: **Hybrid noise inversion with InvSR.** Slerp blending ($\alpha = 0.8$ DDIM, 0.2 InvSR) consistently improves quality.

	2048×2048			4096×4096		
Config	FID ↓	IS ↑	Time	FID ↓	IS ↑	Time
InvSR (only)	65.32	13.22	-	76.87	12.78	-
UltraFlash	50.02	14.03	**2.78**	52.38	13.99	**8.04**
+ InvSR	**49.33**	**14.24**	5.12	**51.34**	**14.04**	13.76

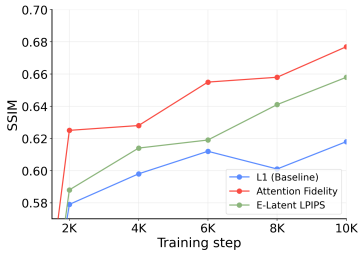
Variant	**FID ↓**	**IS ↑**	**Latency (s) ↓**
PixelRush (SD-Turbo)	56.50	13.42	16.37
+ Latent Hyperloop*	59.18	13.12	11.53
+ Latent Hyperloop	**51.99**	13.89	11.83
+ Sparse Decoding	52.38	**13.99**	**8.04**

* excludes the context-aware patch-based mechanism.

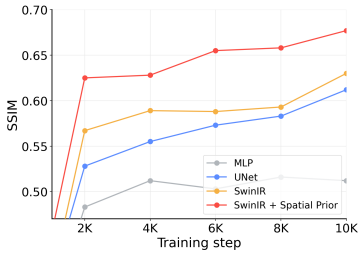
(a) Component-wise Ablation on 4K Generation

Base Generator	**Refinement Model**	**FID ↓**	**IS ↑**
SDXL [20]	SD-Turbo [24]	52.38	13.99
FLUX [12]	SD-Turbo [24]	51.75	**16.61**
SDXL [20]	SDXL-Turbo [24]	<u>51.32</u>	13.93
FLUX [12]	SDXL-Turbo [24]	**50.87**	<u>16.52</u>
SDXL [20]	PixArt-Σ [3]	53.47	13.70

(b) Robustness Across Backbone Combinations



(c) Objective Convergence



(d) Architecture Convergence

Figure A1: **Latent Hyperloop ablations and convergence analysis. Left:** We report 4K generation ablations and robustness across different base/refinement backbone combinations. Latent Hyperloop improves generation quality and latency, while Sparse Patch-based Decoding further reduces inference time with only a small FID trade-off. **Right:** SSIM convergence curves. (c) Attention-based objectives accelerate convergence and improve the final reconstruction quality over the ℓ_1 baseline and E-LatentLPIPS [11]; (d) the Spatial Prior architecture consistently outperforms alternative designs.

latency but causes a clear drop in fidelity (FID degrades from 56.50 to 59.18) due to lack of boundary context. However, by incorporating our ***context-expanded mechanism***, the full Latent Hyperloop not only resolves these artifacts but improves FID to 51.99 while reducing latency by about 28% relative to the baseline (16.37s → 11.83s). Subsequently, **Sparse Patch-based Decoding** provides additive efficiency gains by reducing redundancy in the decoding and sampling stages. Together, the two components reduce the total inference time to 8.04s – a **2**× acceleration compared to PixelRush (16.37s) – while improving visual quality.

Robustness to Model Configurations. To assess the versatility of UltraFlash, we evaluate its performance across diverse settings, varying both the base model and the refinement module. As shown in Fig. A1b, UltraFlash consistently produces strong FID and IS scores regardless of the underlying model.

Latent Hyperloop Design. Fig. A1c–d analyze the design choices behind **Latent Hyperloop**. In Fig. A1c, Attention Alignment improves convergence and final decoded-image fidelity over both the ℓ_1 baseline and E-LatentLPIPS [11], a perceptual loss computed in latent space, showing that refiner attention provides useful structure-aware supervision beyond latent reconstruction alone. In Fig. A1d, **Spatial Prior** further improves SSIM by reusing spatial expansion behavior already present in the VAE decoder. Together, these ablations justify the final **Latent Hyperloop** design.

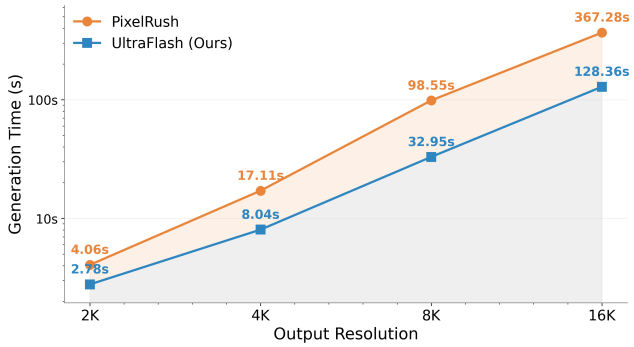


Figure A2: **Scalability Analysis.** Comparison of inference latency across increasing resolutions (log scale). UltraFlash (blue) offers significantly better scaling properties than PixelRush (orange).

Context Size. Table A3 varies the number of context pixels p added on each side of the 64×64 source crop, i.e., the input side length $s + 2p$ of Latent Hyperloop (Sec. 4.1). Quality improves sharply as context is added, with FID decreasing from 59.18 at $p = 0$ to 51.99 at $p = 8$, after which the returns diminish: increasing p to 12 or 16 changes FID by less than 0.35. We therefore use $p = 8$ as the trade-off between quality and cost, since a larger context adds computation for marginal gain.

Table A3: **Ablation on the context size p of Latent Hyperloop.** Same 4K setting as Fig. A1a (SD-Turbo refiner, before adding Sparse Patch-based Decoding); $p = 0$ corresponds to the variant without context expansion.

Context pixels p	0	4	8 (ours)	12	16
FID ↓	59.18	55.86	51.99	51.67	51.71
IS ↑	13.12	13.48	13.89	13.82	13.89

C Additional Qualitative Results

C.1 Latent Hyperloop vs. Image Bilinear Interpolation

Fig. A3 compares the output of our Latent Hyperloop adapter against the standard pixel-space pipeline (VAE decode → bilinear interpolation → VAE encode). For each input latent, we decode both versions to pixel space. The Latent Hyperloop closely matches the pixel-space ground truth across diverse content types.

C.2 Effect of Training Objectives on Latent Hyperloop Reconstruction

Fig. A4 provides a qualitative comparison of the training objectives used for learning the Latent Hyperloop adapter. Using only the ℓ_1 reconstruction loss preserves the coarse object layout, but the reconstructed image remains visibly over-smoothed, with weakened local contrast and softened facial structure. Adding E-LatentLPIPS yields only limited perceptual improvement over ℓ_1 alone. In contrast, incorporating the proposed Attention Alignment objective produces substantially sharper and more faithful reconstructions, especially in fine-grained regions such as the eyes, nose contour, and surrounding fur. These visual differences support the design choice of using Attention Alignment to improve latent-space upsampling quality.

C.3 Additional UltraFlash Outputs

Figs. A5 and A6 present additional 4096×4096 outputs using FLUX as the base model, while Figs. A7 and A8 show results using SDXL as the base model. All images are generated with UltraFlash refinement on a single A100 GPU, demonstrating that our pipeline generalizes across different base architectures.



Figure A3: **Latent Hyperloop vs. image bilinear interpolation.** Our Latent Hyperloop produces outputs that are visually indistinguishable from the expensive pixel-space round-trip. Best viewed **zoomed-in**.

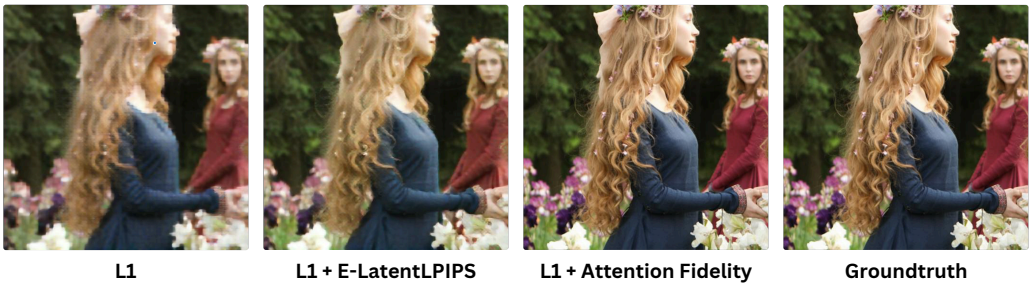


Figure A4: **Effect of training objectives on Latent Hyperloop reconstruction.** From left to right: reconstructions trained with ℓ_1 only, ℓ_1 + E-LatentLPIPS, and ℓ_1 + Attention Alignment, followed by the ground-truth reconstruction. All variants use the same latent upsampling backbone and are trained under the same setting. While ℓ_1 preserves the overall structure, it produces noticeably smooth reconstructions, and E-LatentLPIPS provides only marginal visual improvement. The proposed Attention Alignment objective better preserves local structure and fine facial details, yielding reconstructions that are visually closest to the ground truth. Best viewed zoomed in.

D Limitations

UltraFlash is developed and evaluated for cascade-based megapixel image synthesis. This focus allows us to isolate and optimize a concrete bottleneck that appears in practical high-resolution text-to-image pipelines: VAE-heavy transitions between latent and pixel space. We therefore view the method as a targeted acceleration strategy for image synthesis rather than a claim about all generative settings. Other modalities, such as video generation, introduce additional temporal and architectural considerations that are outside the scope of this work. At the same time, the central observation

behind UltraFlash is broader: as denoising becomes faster, the surrounding transition costs become increasingly important. Exploring transition-aware acceleration beyond image cascades is a natural direction for future work.

E Broader Impact

UltraFlash improves the efficiency of high-resolution image synthesis by reducing transition overhead in cascade pipelines. This can make megapixel generation more practical for creative tools, design workflows, data visualization, and research settings where high-resolution outputs are useful but computational cost is a barrier. At the same time, faster and higher-resolution generation should be deployed with the same safeguards used for the underlying generative models, including content filtering, provenance or watermarking mechanisms when available, and responsible-use policies. UltraFlash does not introduce a new generative prior or bypass the safety behavior of the base models; it accelerates the transition operations around existing cascade refinement. We therefore view its impact as primarily improving efficiency, while inheriting the broader benefits and risks of the image-generation systems on which it is built.

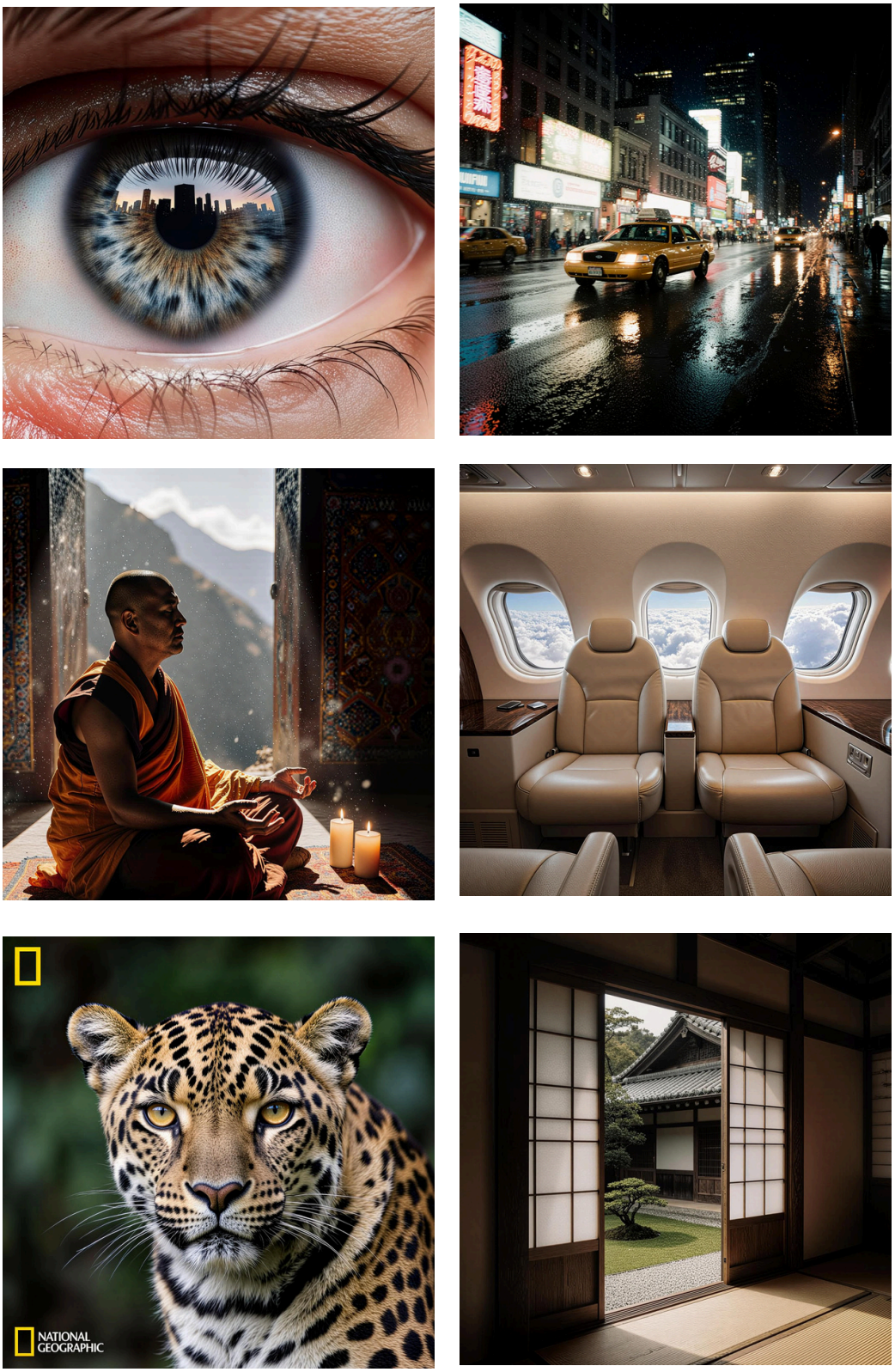


Figure A5: **Uncurated UltraFlash outputs at** 4096×4096**.** Generated with FLUX base model and UltraFlash refinement pipeline.

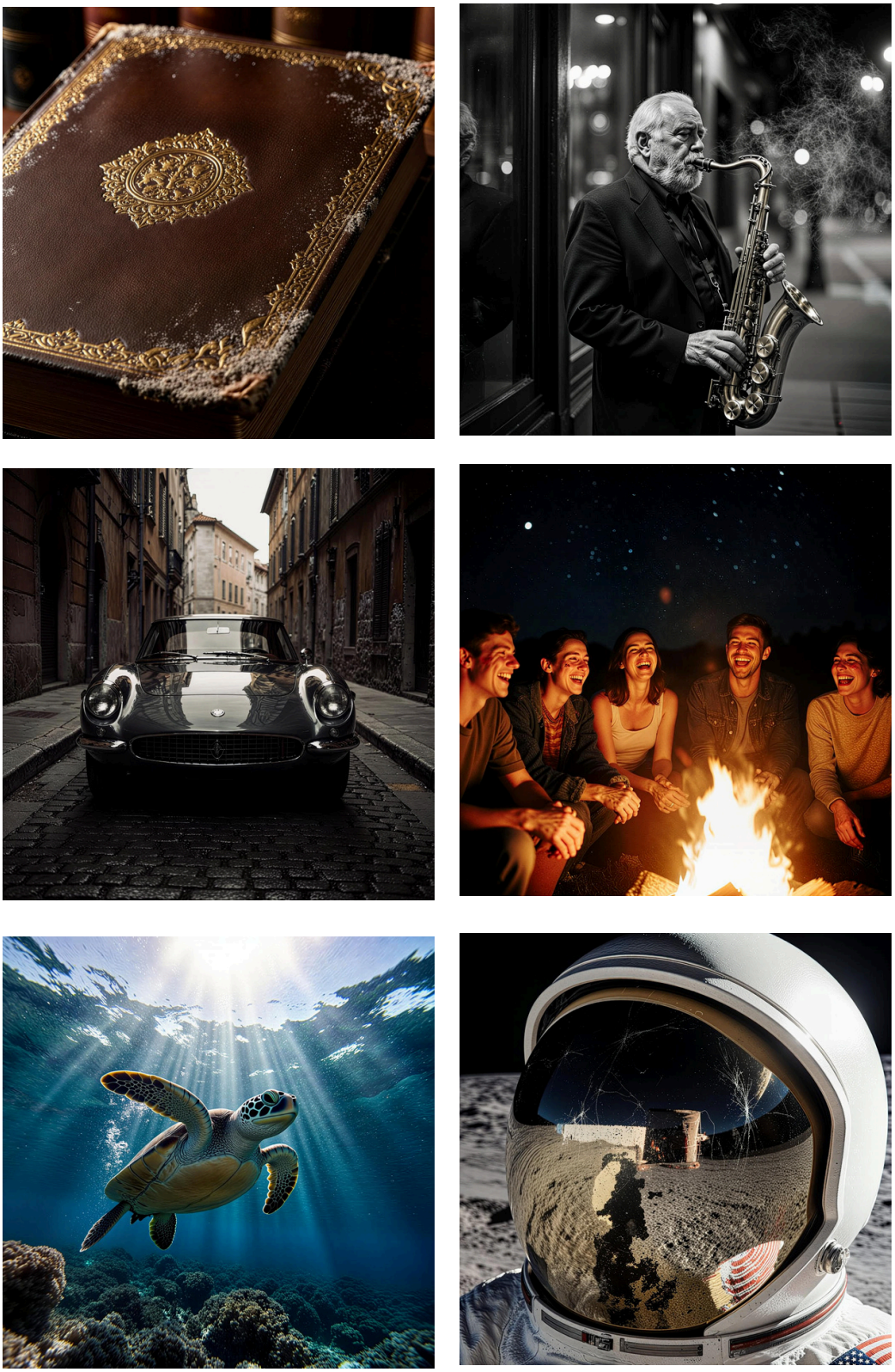


Figure A6: **Uncurated UltraFlash outputs at** 4096×4096**.** Generated with FLUX base model and UltraFlash refinement pipeline.



Figure A7: **Uncurated UltraFlash outputs at** 4096×4096**.** Generated with SDXL base model and UltraFlash refinement pipeline.

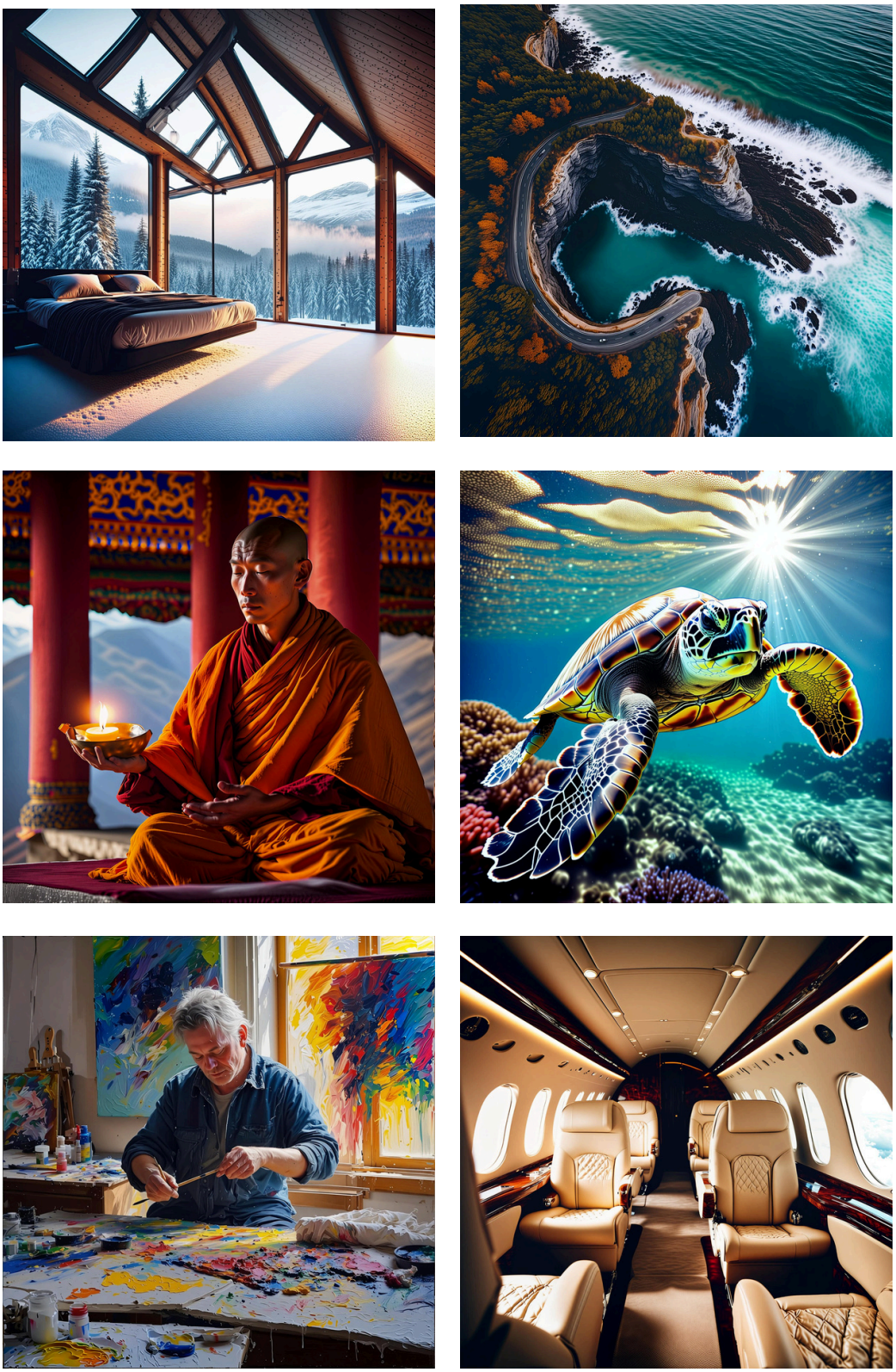


Figure A8: **Uncurated UltraFlash outputs at** 4096×4096**.** Generated with SDXL base model and UltraFlash refinement pipeline.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: The abstract and introduction state the paper's scope as transition-efficient megapixel synthesis, with contributions including Latent Hyperloop, Sparse Patch-based Decoding, and empirical evaluation in cascade pipelines. The claims are supported by the experiments in Section 5, and comparisons with native 4K and FLUX-based methods are explicitly framed as contextual because they use different backbones or training assumptions.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: The paper discusses limitations in Appendix D. The discussion frames the scope of UltraFlash as cascade-based megapixel image synthesis and notes that extending transition-aware acceleration beyond image cascades is a natural direction for future work.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A].

   Justification: The paper does not present formal theoretical results or theorems. The mathematical expressions are used to define the cascade transition operators and training objectives.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes].

   Justification: Section 5 describes the base models, refinement models, datasets/prompts, metrics, resolutions, hardware, and baseline settings. The method section and supplemental material specify the Latent Hyperloop architecture, training objective, implementation details, and Sparse Patch-based Decoding procedure needed to reproduce the main algorithmic results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No].

   Justification: The current submission does not include public code or released checkpoints. We plan to release code and trained adapters upon acceptance.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes].

   Justification: The experimental section reports the evaluation prompts, resolutions, metrics, hardware, base/refinement models, and main baseline configurations. The method and supplemental material describe Latent Hyperloop training data, architecture, losses, optimizer, training steps, and decoding settings.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No].

Justification: We do not report error bars because each full 4K evaluation is computationally expensive and uses 1000 prompts. To reduce variance, all quantitative comparisons use the same prompt set, metrics, hardware, and evaluation protocol.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes].

   Justification: The experimental section reports that inference experiments are run on a single NVIDIA A100-40GB GPU and provides wall-clock runtime for each method. The paper also reports model size for Latent Hyperloop and the inference time needed to reproduce the main speed claims.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: The work uses existing pretrained image-generation models and public datasets/prompts for research evaluation. We do not collect sensitive personal data or deploy the system, and we discuss potential misuse and broader impacts of improved high-resolution generation in Appendix E.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes].

    Justification: The paper discusses broader impacts in Appendix E, including efficiency benefits for high-resolution image synthesis and the need to retain safeguards such as content filtering, provenance or watermarking mechanisms when available, and responsible-use policies inherited from the underlying generative models.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [Yes].

    Justification: The proposed modules are lightweight transition operators rather than standalone image generators. As discussed in Appendix E, UltraFlash does not introduce a new generative prior or bypass the safety behavior of the underlying generative models; it should be used with the same content filtering, provenance or watermarking mechanisms when available, and responsible-use policies as the base models. If released, the code and adapters will be distributed for research use with documentation, and no new scraped image dataset is released.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes].

    Justification: The paper cites the pretrained models, datasets, and baseline methods used in the experiments, including SDXL, SDXL-Turbo, SD-Turbo, JourneyDB, LAION-Aesthetics, FouriScale, DemoFusion, FreeScale, and PixelRush. We use these assets according to their intended research/evaluation settings and provide references for attribution.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A].

    Justification: The paper does not introduce a new dataset or benchmark. The proposed Latent Hyperloop adapter and decoding procedure are documented in the method and supplemental material; if released, the trained adapters and code will include documentation and usage instructions.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [No].

    Justification: The user preference study was an unpaid, voluntary perceptual evaluation. Participants were presented with side-by-side generated images and asked to select the one with better overall quality, text alignment, and structure. Because no commercial crowdsourcing platform was used and no financial compensation was provided, platform and compensation details are not applicable; the current submission does not include full interface screenshots.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [No].

    Justification: The user study is a low-risk anonymous preference evaluation of generated images, but the current submission does not describe IRB or equivalent review. We will add details on participant risk, consent, anonymity, and institutional requirements in the supplemental material.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A].

    Justification: The core method does not use LLMs as an algorithmic component. Any language-model assistance, if used, was limited to writing, editing, or formatting and did not affect the scientific method, experiments, or originality of the work.