EXPLORING THE VAE ENCODER DESIGN SPACE FOR LATENT DIFFUSION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Latent diffusion models often retain VAE encoder architectures introduced by early LDM systems, despite significant evolution in both decoders and diffusion backbones. We revisit the encoder from an efficiency perspective, and study how the computation should be allocated with regard to operations and resolution stages. We introduce **ABCRP** and **LatentMaid**: encoders built from pixel unshuffle, RMSNorm, and a depthwise + SwiGLU residual block. The design keeps spatial effects in the decoder and uses only local, lightweight components in the encoder, explicitly restricting cross-spatial dependencies in the encoder without requiring an auxiliary locality loss. At CFG=1, ABCRP with standard residual blocks achieves the best generation quality in our study (17.68 gFID), while LatentMaid achieves 18.32 gFID with 89% fewer encoder FLOPs than the SD-VAE baseline and increasing measured encoder throughput by approximately 3×, providing a substantially more efficient operating point.

All code has been uploaded as supplementary material - available for testing and reproduction.

1 INTRODUCTION

Latent diffusion models Rombach et al. (2022) have become a dominant paradigm for high-quality image synthesis. Their two-stage design—an autoencoder compresses images into latents, followed by a generative backbone Ho et al. (2020); Peebles & Xie (2023); Esser et al. (2024); Black Forest Labs (2024) that models the latent distribution—greatly mitigates the computational cost associated with pixel-space resolutions Dhariwal & Nichol (2021); Crowson et al. (2024); Hoogeboom et al. (2023), enabling the widespread adoption of diffusion generative models. Over time, diffusion backbones have evolved from U-Nets Ronneberger et al. (2015) to transformers Vaswani et al. (2017), but the autoencoder architecture has largely followed a fixed protocol: the KL-regularized convolutional VAE Kingma & Welling (2013) remains virtually unchanged, from LDM Rombach et al. (2022) and SDXL Podell et al. (2023) to Stable Diffusion 3 Esser et al. (2024) and FLUX Black Forest Labs (2024). Meanwhile, the performance of VAEs is mainly evaluated via reconstruction metrics such as rFID Heusel et al. (2017) and PSNR. However, recent studies suggest that reconstruction quality is misaligned with generative fidelity (gFID) Skorokhodov et al. (2025); Chen et al. (2025d); Yao et al. (2025b). In other words, an autoencoder with superior reconstruction capability may produce latent representations that are poorly suited for the subsequent diffusion process, leading to degraded sample quality, such as blurry structures, distorted geometry, or texture artifacts.

Prior observations suggest that latent smoothness and receptive fields matter across settings Guo et al. (2024); Siméoni et al. (2025); Bradbury & Zhong (2026), with some works Siméoni et al. (2025) demonstrating that as training scales and progresses, global information starts leaking into all latents, hindering dense tasks. Remedies often include solutions such as auxiliary losses, secondary networks. These finding motivate studying encoders with reduced token mixing. Rather than penalizing token mixing through auxiliary loss or network, we ask whether related spatial constraints can be imposed by construction: standard encoders introduce cross-spatial coupling through strided convolutions, GroupNorm Wu & He (2018), and self-attention Vaswani et al. (2017), and replacing each with a spatially-local alternative. Whereas these approaches penalize latent properties after they appear, LatentMaid enforces locality and thus efficiency through encoder architecture. The ABCRP chain uses a lossless pixel unshuffle Shi et al. (2016) followed by a learned pointwise

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

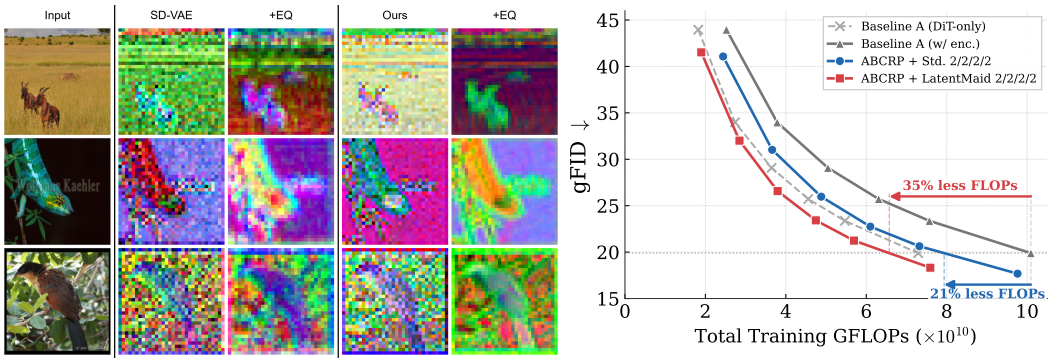


Figure 1: **Latent Structure (Left)** Top three principal components of SD-VAE and LatentMaid 2/2/2/2 latents, with and without EQ-VAE; LatentMaid yields smooth latents, while continuing to benefit from EQ. **Accelerated Training (Right)** Training curves of gFID vs total training FLOPs; LatentMaid matches the SD-VAE baseline with **89%** fewer encoder FLOPs and **35%** fewer total DiT training FLOPs with noncached latents. Using cached latents and including VAE training yields **19%** savings instead.

projection for downsampling, GroupNorm-to-RMSNorm Zhang & Sennrich (2019) and SiLU-to-Mish Misra (2019) substitutions for local normalization, and removal of encoder self-attention for bounded spatial interaction. The LatentMaid block further factorizes local depthwise spatial processing from SwiGLU channel mixing, retaining channel capacity while bounding the spatial radius of learned operations.

In summary, our contributions are as follows:

1. We study locality-preserving encoder design for latent diffusion through the progressive **A** → **AB** → **ABC** → **ABCR** → **ABCRP** architecture chain, in which each step replaces a component (self-attention, GroupNorm, strided convolution) with a more spatially-local alternative. Within the chain we identify an encoder/decoder asymmetry around self-attention: removing it from the encoder sometimes improves gFID, while removing it from the decoder degrades both reconstruction and generation - we focus our efforts on the encoder as a result.
2. We propose **LatentMaid**, a locality-preserving VAE architecture that replaces each mixing-prone component with an information-preserving alternative, achieving a gFID of **18.32** (CFG=1, 400k DiT training steps) with **89%** fewer encoder FLOPs and establishing an internal Pareto frontier for the efficiency–generation quality trade-off.
3. We propose **ABCRP**, a localized, small encoder, achieving our best gFID of **17.68** (CFG=1, 400k DiT training steps) with fewer encoder FLOPs than the baseline,

2 Related Work

Latent diffusion models. Latent diffusion models Rombach et al. (2022) have established a dominant two-stage paradigm for high-quality image synthesis: an autoencoder first compresses images into a lower-dimensional latent space, and a generative backbone, originally a U-Net Ho et al. (2020), later replaced by transformers Peebles & Xie (2023); Ma et al. (2024) and rectified-flow Liu et al. (2022) formulations Esser et al. (2024); Black Forest Labs (2024), learns the latent distribution. Subsequent improvements to diffusion modeling and sampling Nichol & Dhariwal (2021); Song et al. (2021) have primarily targeted the generative process itself rather than the encoder interface. Despite rapid evolution of the generative backbone, the autoencoder architecture has remained largely unchanged since the original KL-regularized convolutional VAE introduced alongside LDM Rombach et al. (2022). Subsequent systems such as SDXL Podell et al. (2023), Stable Diffusion 3 Esser et al. (2024), and FLUX Black Forest Labs (2024) all inherit essentially the same encoder–decoder design, treating the autoencoder as a fixed interface rather than a tunable compo-

nent. This architectural stagnation motivates our systematic study of how encoder design choices affect downstream generation quality.

Autoencoder design for generation. Recent work reveals that reconstruction quality alone poorly predicts generation performance Li et al. (2025); Wang et al. (2025); Cheng & Yuan (2025).

Regularization-driven methods add auxiliary losses without modifying the architecture: EQ-VAE Kouzelis et al. (2025) enforces spatial equivariance, Skorokhodov et al. Skorokhodov et al. (2025) use scale-equivariance regularization, and Diffusion Autoencoders Preechakul et al. (2022) optimize representation structure jointly with a diffusion-style objective. Orthogonal directions include frequency-aware curricula Lai et al. (2025), multi-scale tokenizers Chen et al. (2025a), and training tricks Yao et al. (2025a) such as improved diffusion architecture or sampling schedules. Few of these works trace latent-space structure to specific encoder components Skorokhodov et al. (2025); Chen et al. (2025d); Yao et al. (2025b); Bradbury & Zhong (2026). We instead design a locality-preserving encoder that natively yields smoother latents, drawing on token-mixing in self-supervised vision Siméoni et al. (2025). Our approach is complementary to these directions and stacks with regularization and compression; alignment losses are a natural extension.

3 PRELIMINARIES

3.1 VARIATIONAL AUTOENCODERS FOR LATENT GENERATIVE MODELING

In latent generative modeling, the VAE defines the interface between pixels and the generative model; its reconstruction, adversarial, and latent-regularization terms each affect the codes later consumed by diffusion.

Let $\mathbf{x} \in \mathbb{R}^{H\times W\times 3}$ denote an input image. A variational autoencoder (VAE) Kingma & Welling (2013) comprises an encoder $\mathcal{E}$ and a decoder $\mathcal{D}$. The encoder maps $\mathbf{x}$ to a latent $\mathbf{z} = \mathcal{E}(\mathbf{x}) \in \mathbb{R}^{\frac{H}{f}\times\frac{W}{f}\times c}$, where f is the spatial downsampling factor and c the number of latent channels. The decoder reconstructs $\hat{\mathbf{x}} = \mathcal{D}(\mathbf{z})$. The standard training objective reads:

$$\mathcal{L}_{\text{VAE}}(\mathbf{x}) = \mathcal{L}_{\text{rec}}(\mathbf{x}, \hat{\mathbf{x}}) + \lambda_{\text{gan}}\mathcal{L}_{\text{gan}}(\hat{\mathbf{x}}) + \lambda_{\text{reg}}\mathcal{L}_{\text{reg}}(\mathbf{z}), \tag{1}$$

where $\mathcal{L}_{\text{rec}}$ combines a pixel reconstruction and a perceptual loss Zhang et al. (2018), $\mathcal{L}_{\text{gan}}$ is a patch-based adversarial loss Isola et al. (2017), and $\mathcal{L}_{\text{reg}}$ is a KL divergence toward a standard Gaussian prior Kingma & Welling (2013).

3.2 LATENT DIFFUSION MODELS

Latent diffusion models (LDM) Rombach et al. (2022) trade pixel-space denoising for denoising in a compressed representation, reducing compute while making the autoencoder's latent geometry part of the generative problem.

LDMs decompose generation into two stages Ho et al. (2020); Dhariwal & Nichol (2021). In the first, a VAE projects the input image $\mathbf{x}$ into a latent $\mathbf{z} = \mathcal{E}(\mathbf{x})$, discarding perceptually-insignificant high-frequency detail. In the second, a diffusion model approximates the latent data distribution $q(\mathbf{z})$ by iteratively denoising Gaussian noise, optimised via the standard variational objective Ho et al. (2020):

$$\mathcal{L}_{\text{LDM}} = \mathbb{E}_{\mathbf{z},\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I}),t}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t)\right\|^2\right], \tag{2}$$

where $\boldsymbol{\epsilon}_\theta$ predicts the noise added at timestep t and $\mathbf{z}_t$ is the latent corrupted with Gaussian noise at level t. At inference, the model is iteratively denoised to a sample $\mathbf{z}_0$ and decoded as $\hat{\mathbf{x}} = \mathcal{D}(\mathbf{z}_0)$.

3.3 CONTROLLING SPATIAL DEPENDENCIES IN LATENT ENCODERS

We investigate two related encoder properties: the spatial extent of local feature aggregation and the presence of operations that introduce global spatial dependencies. Our aim is to limit both, while retaining sufficient spatial context for reconstruction.

Let $\mathbf{z}_i = E(\mathbf{x})_i$ denote the latent token at spatial position i. We define its theoretical receptive field $\mathcal{R}_i$ as the set of input positions that can influence $\mathbf{z}_i$. Thus,

$$\mathbf{z}_i = g_i(\mathbf{x}_{\mathcal{R}_i}).$$

For an encoder composed of spatially local operations, the extent of $\mathcal{R}_i$ is determined by the composition of convolution kernels, strides, and spatial rearrangements. Neighboring latent tokens may have overlapping receptive fields; locality does not imply independent or non-overlapping token representations.

Some operations introduce dependencies beyond this local aggregation structure. Group normalization Wu & He (2018) computes statistics across spatial positions within each channel group, coupling distant locations through shared statistics. Global self-attention Vaswani et al. (2017) instead allows content-dependent aggregation across all spatial positions. Both can make the theoretical receptive field cover the entire input, although they differ in the form and strength of the dependencies they introduce.

Our encoder modifications address these properties separately. Removing self-attention eliminates explicit global attention-based aggregation, while replacing group normalization with channel-wise RMS normalization removes spatial coupling through normalization statistics. Changes to the local encoder architecture, including pixel-unshuffle downsampling and revised residual blocks, alter the remaining receptive-field extent. Pixel unshuffle itself is a spatial-to-channel rearrangement; subsequent channel mixing and spatial convolutions determine how the rearranged features are combined. Pointwise activation changes do not alter theoretical spatial support.

These modifications provide architectural control over spatial dependencies. They do not by themselves establish improved latent disentanglement or generation quality, which we assess empirically.

4 METHOD

Starting from the standard LDM encoder Rombach et al. (2022), we systematically replace each mixing-prone component with a locality-preserving alternative (Sec. 4.2), introduce a block design that factorizes spatial and channel processing (Sec. 4.3), and concentrate learned computation at the stages where it most benefits generation (Sec. 4.4). The decoder and training objective remain unchanged throughout, so all improvements are attributable to encoder design alone. This focus isolates encoder-side effects; DC-AE 1.5 Chen et al. (2025d) applies its adversarial signal only to the decoder's final layer, leaving decoder–latent interactions for separate study.

4.1 BASELINE ARCHITECTURE

We first establish the baseline that all subsequent modifications build upon. Our starting point is the convolutional encoder from the standard latent-diffusion VAE Rombach et al. (2022), inherited from the Taming Transformers first-stage design Esser et al. (2021). The encoder comprises four resolution stages with block counts $2/2/2/4$ and channel multipliers $1/2/4/4$ relative to a base width of 128. Each stage stacks residual blocks—two 3×3 convolutions with GroupNorm Wu & He (2018) and SiLU activations—followed by a strided-convolution downsampling layer; a mid-block with self-attention sits between the encoder trunk and the latent projection. Only the first 3 stages perform downsampling. The resulting latent $\mathbf{z} \in \mathbb{R}^{\frac{H}{f} \times \frac{W}{f} \times c}$ has spatial factor $f{=}8$ and $c{=}4$ channels. We denote this configuration as **A**.

Three components in this architecture introduce cross-spatial coupling: self-attention mixes all tokens globally, GroupNorm aggregates statistics across spatial positions, and strided convolutions aggregate overlapping local neighborhoods during downsampling. We address each source in turn.

4.2 LOCALITY-PRESERVING ENCODER MODERNIZATION

We adopt a simple design rule: encoder operations are restricted to spatially-local interactions, improving efficiency and causing minimal change to downstream results. We cumulatively replace each mixing-prone component (self-attention, GroupNorm, and strided convolution) with a locality-preserving alternative, yielding the progression **A** $\rightarrow$ **AB** $\rightarrow$ **ABC** $\rightarrow$ **ABCR** $\rightarrow$ **ABCRP** for

controlled ablation in Sec. 5. All modifications target the encoder only, motivated by our experiments (Sec. 5.2): token mixing plays asymmetric roles across the bottleneck, sometimes hurting downstream generation in the encoder while benefiting reconstruction (and, indirectly, generation) in the decoder.

Removing self-attention. Self-attention is the densest source of global token coupling in the encoder: the mid-block computes pairwise affinities across all N^2 token pairs, creating direct long-range dependencies between latent positions. We remove encoder self-attention (configuration **AB**), while retaining it in the decoder (Sec. 5.2). Tab. 1 confirms the asymmetry: removing encoder attention alone improves generation quality, whereas additionally removing decoder attention degrades it.

Replacing spatial normalization. GroupNorm Wu & He (2018) introduces implicit inter-token coupling by computing mean and variance across all spatial positions within each channel group. This dependency persists even when convolutions are strictly local; loss-based regularisation Kouzelis et al. (2025) can discourage it without removing its architectural source. To make normalization position-independent, we replace GroupNorm with RMSNorm Zhang & Sennrich (2019), applied per position (configuration **ABCR**):

$$\mathrm{RMSNorm}(\mathbf{h}_i) = \frac{\mathbf{h}_i}{\sqrt{\frac{1}{d}\sum_{j=1}^{d} h_{ij}^2 + \epsilon}} \odot \boldsymbol{\gamma}, \tag{3}$$

where $\mathbf{h}_i \in \mathbb{R}^d$ is the feature vector at position i and $\boldsymbol{\gamma} \in \mathbb{R}^d$ is a learned scale. Because statistics are computed per token, no information flows between spatial positions through the normalization path.

We additionally substitute SiLU with Mish Misra (2019) ($\mathrm{Mish}(x)=x\cdot\tanh(\mathrm{softplus}(x))$), which yields a slightly softer optimization landscape (configuration **ABC**).

Locality-preserving downsampling. Strided convolutions aggregate overlapping input neighborhoods during downsampling, causing neighboring output tokens to depend on overlapping spatial regions. We instead replace the downsampling with pixel unshuffle Shi et al. (2016) followed by a 1×1 projection (configuration **ABCRP**):

$$\mathbf{y} = W_{1\times1}\,\mathrm{PixelUnshuffle}_2(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^{H\times W\times C} \to \mathbf{y} \in \mathbb{R}^{\frac{H}{2}\times\frac{W}{2}\times C'}. \tag{4}$$

Pixel unshuffle partitions the input into non-overlapping 2×2 regions and rearranges their spatial offsets into channels. The subsequent projection can learn within-region mixing, but the downsampling operation does not mix information across distinct output spatial locations. For consistency purposes, we also replace the input convolution with pixel unshuffle.

The full modernization stack is denoted **ABCRP**.

In Appendix G.1 we document the exact changes to locality and receptive field as a function of these changes.

4.3 LatentMaid Block

The ABCRP modernization eliminates global token mixing, but a subtler issue remains: within each residual block, two stacked 3×3 full convolutions simultaneously perform spatial aggregation and channel transformation, preventing independent control of the spatial mixing budget. We aim to decouple these two functions so that spatial locality can be enforced with minimal loss in representational capacity.

To this end, we factorize each block into two specialized operations: a depthwise convolution for spatially local aggregation (per-channel, bounded kernel) and a pointwise gated activation for channel transformation (per-position, spatially agnostic). This yields the **LatentMaid block** (Fig. 2):

$$\mathbf{h}' = \mathbf{h} + \mathrm{SwiGLU}\big(\mathrm{DWConv}(\mathrm{RMSNorm}(\mathbf{h}))\big), \tag{5}$$

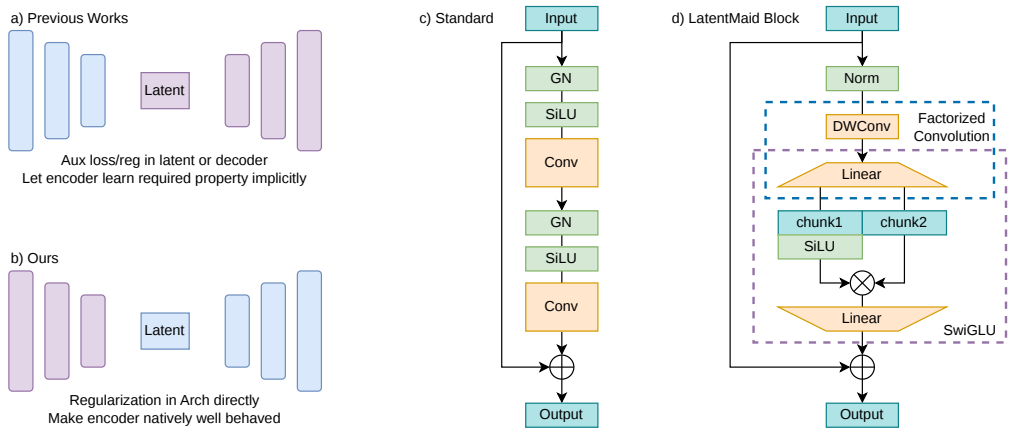


Figure 2: **Enforcing encoder locality by construction** (a) Prior work regularizes the latent or decoder via an auxiliary loss; (b) Ours: restrict specified encoder dependencies architecturally (c) Standard residual block (GN+SiLU+3×3 Conv, twice). (d) LatentMaid block: RMSNorm, depthwise convolution (spatial), and SwiGLU (channel); dashed boxes mark the factorised-convolution and SwiGLU interpretations.

where the depthwise convolution controls strictly local spatial mixing and SwiGLU Shazeer (2020) provides expressive channel processing through gating. Compared with a standard two-layer residual block of the same channel width, this factorized design (i) explicitly bounds the spatial mixing radius to the depthwise kernel size, making token interaction controllable; (ii) reduces both parameters and FLOPs substantially (Sec. 5.3); and (iii) preserves representational capacity through gated channel processing without requiring spatial expansion.

Factorized-convolution interpretation. The depthwise convolution followed by the linear input projection of SwiGLU is equivalent to a single dense convolution with a rank-constrained kernel. In index notation, the effective kernel factorizes as

$$\left(\mathcal{W}_{\text{fact}}\right)_{iouv} = \left(\mathcal{W}_{\text{dw}}\right)_{i1uv}\left(W_{\text{pw}}\right)_{io}, \tag{6}$$

where $\mathcal{W}_{\text{dw}}$ is the depthwise kernel (spatial-only, per-channel) and W_{pw} is the pointwise projection (channel-only, spatial-agnostic). The block thus implements: factorized convolutional input, gated nonlinear channel mixing, linear output projection, and a residual connection. See I for more details.

4.4 Stage Allocation

Not all resolution stages contribute equally to downstream generation. The standard encoder distributes blocks as $2/2/2/4$, devoting the majority of computation to high-resolution stages where each token covers only a small image patch with minimal semantic content. Under locality-preserving design, learned blocks at these early stages expand the effective receptive field over fine-grained tokens—introducing spatial mixing without proportional benefit for generation. We therefore concentrate computation at the stages where tokens carry the most semantic information by progressively removing blocks from the highest-resolution stages:

$$2/2/2/4 \;\rightarrow\; 2/2/2/2 \;\rightarrow\; 0/2/2/2 \;\rightarrow\; 0/0/2/2 \;\rightarrow\; 0/0/0/2. \tag{7}$$

When a stage has zero blocks, only pixel-unshuffle-plus-1×1-projection is applied. This acts as structured, non-overlapping patchification with learned within-patch channel projection, but no learned filtering across output spatial locations. This yields two benefits: computation is focused at lower-resolution stages where each token spans a semantically meaningful image region, and the encoder becomes substantially more efficient in both parameters and throughput (Sec. 5.3). We apply this sweep to both the standard residual block and the LatentMaid block under ABCRP, yielding two complementary families that trade encoder capacity for efficiency (Sec. 5).

4.5 Training Objective

To isolate the contribution of architectural design, we keep the training objective identical to the standard latent-diffusion setup:

$$\mathcal{L}_{\text{AE}} = \mathcal{L}_{\text{rec}} + \lambda_{\text{perc}}\mathcal{L}_{\text{perc}} + \lambda_{\text{gan}}\mathcal{L}_{\text{gan}} + \lambda_{\text{kl}}\mathcal{L}_{\text{kl}}, \tag{8}$$

combining pixel-space reconstruction, LPIPS perceptual loss Zhang et al. (2018), patch-based adversarial training Isola et al. (2017), and KL regularization. No auxiliary equivariance, alignment, or frequency-domain losses are introduced. This demonstrates that the resulting architecture is a complementary to penalizing Kouzelis et al. (2025): architectural modification demonstrates reduced training costs, and minimal degradation/improvement. After autoencoder training, the diffusion backbone is trained on frozen latents for downstream evaluation (Sec. 5).

Table 1: DiT-XL/2 downstream training, VAEs have decoder attention

Config	rFID ↓	PSNR ↑	gFID ↓
A / Baseline	1.39	25.21	19.89
AB w/ dec. attn	0.948	25.79	17.92
ABCRP w/ dec. attn	0.632	**26.23**	19.17
ABCRP 2/2/2/2 w/ dec. attn	0.696	26.13	**17.68**

5 Experiments

5.1 Experimental Setup

Pipeline. We follow the standard two-stage latent-diffusion evaluation protocol. Each autoencoder variant is trained on the ImageNet-1k Deng et al. (2009) training split for up to 250k steps with batch size 64; adversarial training is enabled after 50k steps, matching the original LDM setup Rombach et al. (2022). For downstream evaluation, we select the final checkpoint and train a class-conditional DiT Peebles & Xie (2023) model for 400k steps with batch size 256.

Model variants. Using the cumulative notation from Sec. 4, we evaluate configurations **A** through **ABCRP**, each adding one locality-preserving modification. Under the full ABCRP modernization, we compare the standard residual block with the proposed LatentMaid block across five stage allocations: $2/2/2/4$, $2/2/2/2$, $0/2/2/2$, $0/0/2/2$, and $0/0/0/2$. All modifications are encoder-only; the decoder retains the original architecture throughout unless otherwise noted.

Metrics and Protocol We report reconstruction FID (rFID) and generation FID (gFID), both computed on the full ImageNet-1k validation set (50,000 images). All gFID values are measured without classifier-free guidance unless specified otherwise, on 50,000 images with a consistent seed. We use 250 Euler steps. We additionally report PSNR and encoder parameters/FLOPs where relevant. See Appendix C for further evaluation details.

Baselines. Our primary baseline is the unmodified SD-VAE architecture, (config **A**, gFID 19.89), trained and evaluated under the same protocol as all other configurations.

5.2 Ablation Studies

To isolate the effect of each architectural change, we evaluate the cumulative chain **A** → **ABCRP** at the default $2/2/2/4$ stage allocation (Tab. 2) with a DiT-B size model. See table 6 for DiT hyperparameters, see Appendix B for exact training details.

We scale up the downstream model (Tab. 1). The results are promising: **AB** with decoder attention improves gFID from 19.89 to 17.92, while maintaining competitive rFID (0.95). This means that in addition to using less FLOPs and being more local, the model is performant.

Under full modernization, **ABCRP** with decoder attention achieves rFID 0.63, approaching state-of-the-art reconstruction, with gFID 19.17, still outperforming the baseline. At stage allocation

$2/2/2/2$, this configuration reaches gFID 17.68, the best generation quality among all evaluated configurations, despite a higher rFID of 0.70.

Table 2: Cumulative component ablation at fixed stage allocation $2/2/2/4$, DiT-B. Each row adds one modification to the previous configuration.

Config	Modification	gFID with Decoder Attn. ↓	gFID without Decoder Attn. ↓
A	Baseline	40.6784	N/A
AB	− Encoder Attention	41.0653	43.6183
ABC	+ Mish	41.4529	44.4679
ABCR	+ RMSNorm	41.1990	40.6866
ABCRP	+ pixel unshuffle	42.8603	44.9620

In order to demonstrate that our results are not dataset specific - we train DiT-B model on LSUNYu et al. (2015), demonstrated in (Tab. 3).

Table 3: Comparing baseline versus best in class on LSUN-Bedroom 256, DiT-B scale. We train for 400k steps at BS 256. Due to the smaller dataset, we use the VAEs from the Imagenet training.

Config	gFID ↓
A	12.08
ABCRP 2/2/2/2	5.93
LatentMaid 2/2/2/2	7.78

5.3 Encoder Design Space

We sweep stage allocations for both block families under ABCRP, with decoder attention retained throughout (Tab. 4). The $2/2/2/2$ configuration is optimal for both, yielding gFID 17.68 (standard block) and 18.32 (LatentMaid), and the trend is U-shaped: reducing the last stage from four blocks to two improves gFID, while further removing intermediate stages ($0/2/2/2$, $0/0/2/2$) gradually degrades it. The sweep reveals a U-shaped quality–capacity relationship: removing two blocks from the final stage improves gFID, whereas more aggressive removal eventually degrades it. We treat this as an empirical allocation result rather than evidence for a particular causal mechanism.

Table 4: Main results on class-conditional ImageNet 256×256. All models are trained for 400K steps with the normal SiT-XL/2 architecture. VAE decoder retains self-attention. FLOPs are for a encoder forward pass, single image.

Block family	Stage config	Enc Params	FLOPs	rFID ↓	PSNR ↑	gFID ↓
Baseline **A**	$2/2/2/4$	34.2M	273G	1.39	25.21	19.89
AB (no enc. attn)	$2/2/2/4$	33.1M	269G	0.95	25.79	17.92
ABCRP + std. block	$2/2/2/4$	31.4M	260G	0.63	26.23	19.17
ABCRP + std. block	$2/2/2/2$	21.9M	241G	0.70	26.13	17.68
ABCRP + std. block	$0/2/2/2$	21.4M	179G	0.72	25.99	18.27
ABCRP + std. block	$0/0/2/2$	19.1M	135G	0.81	25.42	19.16
ABCRP + std. block	$0/0/0/2$	10.2M	55.6G	0.85	25.93	18.40
ABCRP + LatentMaid	$2/2/2/4$	8.06M	35.8G	0.92	25.79	23.07
ABCRP + LatentMaid	$2/2/2/2$	4.90M	29.4G	0.85	25.97	18.32
ABCRP + LatentMaid	$0/2/2/2$	4.82M	21.7G	0.82	25.95	19.29
ABCRP + LatentMaid	$0/0/2/2$	4.50M	14.1G	0.88	25.90	20.46
ABCRP + LatentMaid	$0/0/0/2$	3.30M	6.67G	1.40	25.74	22.58

At this shared optimum, LatentMaid uses $7\times$ fewer parameters and $9\times$ fewer FLOPs than the SD-VAE baseline while staying within 1 gFID point of the best standard-block result, establishing an

internal Pareto frontier in the low-compute regime (Fig. 3). The benefit propagates to training dynamics (Fig. 1): ABCRP+std. and LatentMaid at $2/2/2/2$ reach the baseline gFID $1.23\times$ and $1.16\times$ faster in training steps, and LatentMaid matches it with 35% fewer total training FLOPs (encoder forward + DiT forward/backward). Complete analysis of training FLOPs is in Appendix H.

ABCRP 2/2/2/2 achieves the best absolute generation quality, with a gFID of 17.68. LatentMaid at the same stage allocation obtains a slightly higher gFID of 18.32, a difference of 0.64, while reducing encoder FLOPs from 241G to 29.4G and encoder parameters from 21.9M to 4.9M. Thus, the standard-block configuration is preferable when generation quality is the primary objective, whereas LatentMaid targets the low-compute operating regime.

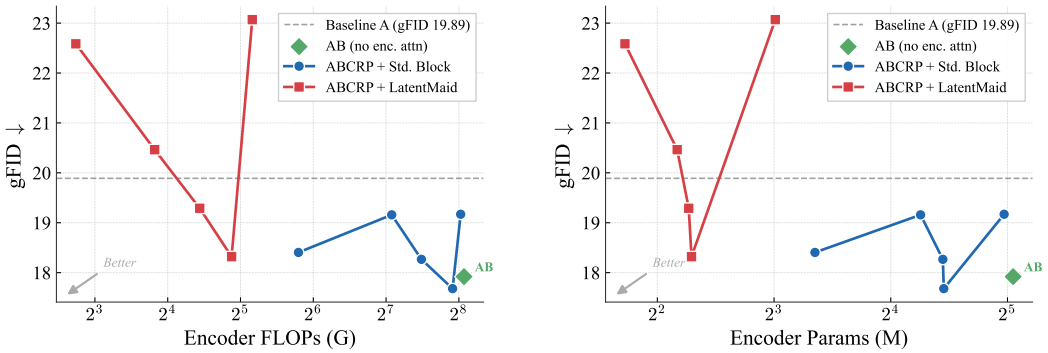


Figure 3: Encoder efficiency vs. generation quality. Left: FLOPs vs. gFID. Right: Params vs. gFID. The dashed line marks the baseline A. LatentMaid occupies the low-compute region while maintaining competitive gFID.

Additional quantitative and qualitative results, including a classifier-free guidance sweep, the $f16c16$ higher-compression regime, a combination with EQ-VAE regularisation, and an rFID vs gFID comparison across all configurations (Tab. 11; Pearson r=0.382 across N=12 configurations), are deferred to Appendix E.

6 Conclusion

We present two different encoder lines: LatentMaid and ABCRP, locality-preserving VAE encoders for latent diffusion. On ImageNet 256×256 with a DiT-XL/2 backbone, LatentMaid attains gFID 18.32 at CFG=1 with 89% fewer encoder FLOPs than the SD-VAE baseline, reaches baseline generation quality with 35% fewer total training FLOPs. These benefits preserve in training throughput, where LatentMaid trains around 32% faster and establishes a promising, internal Pareto frontier for the trade-off between encoder efficiency and generation quality. ABCRP 2222 with Decoder Attention gives marginal benefits in FLOPs, but has 3x the encoding speed, measured on real B200 hardware. This results in end-to-end speed increasing around 30%, despite the unchanged decoder setup. With a gFID of 17.68, this represents measurable improvements.

The encoder and decoder play asymmetric roles around self-attention: removing it from the encoder has marginal effects on gFID, while removing it from the decoder can degrade both reconstruction and generation (Tab. 1); global self-attention continues to play a role in the decoder. Within the encoder, stage allocation follows a U-shape with $2/2/2/2$ achieving strongest results for both the standard residual block and the LatentMaid block - outperforming baselines in both compute and downstream results. And factorizing spatial and channel processing retains competitive generation quality of substantially larger encoders at distinctly lower parameter and FLOP cost, with the resulting latents also accelerating downstream DiT convergence (Fig. 1) beyond the per-step encoder savings.

Taken together, these results suggest that the latent-diffusion encoder need not be the heavyweight component the field has inherited from early VAE pipelines. Treating the encoder as a tunable part of the design space exposes substantial efficiency gains before generation quality is compromised, and remains compatible with existing regularization methods.

AI USE STATEMENT

In this work, we used generative AI tools for feedback on experiment design. We have not used generative AI tools for any other tasks, and the rest of the required disclosure tasks are not applicable to this work. In addition, we used generative AI tools for paper refinement. We have reviewed all AI-assisted work - everything written in the paper was reviewed by humans. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

All image generation systems have potential for misuse, such as dataset biases. We do not introduce a dataset. We do not feel there any special ethical considerations.

REPRODUCIBILITY STATEMENT

We provide code as supplementary material, which allows for the retraining of all models (in conjunction with openly available codebase LightningDiT. All data is publicly available, all hyperparameters for training models are provided. All results are designed to be reproducible using the supplied code and hyperparamters.

REFERENCES

Black Forest Labs. FLUX.1: Rectified flow transformer for high-quality image generation. `https://github.com/black-forest-labs/flux`, 2024.

Rowan Bradbury and Dazhi Zhong. Your latent mask is wrong: Pixel-equivalent latent compositing for diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18630–18639, 2026.

Cong Chen et al. HieraTok: Multi-scale visual tokenizer improves image reconstruction and generation. *arXiv preprint arXiv:2509.23736*, 2025a.

Hao Chen, Yujin Han, Fangyi Chen, Xiang Li, Yidong Wang, Jindong Wang, Ze Wang, Zicheng Liu, Difan Zou, and Bhiksha Raj. Masked autoencoders are effective tokenizers for diffusion models. In *Int. Conf. Mach. Learn.*, 2025b.

Junyu Chen, Han Cai, Junsong Chen, Enze Xie, Shang Yang, Haotian Tang, Muyang Li, Yao Lu, and Song Han. Deep compression autoencoder for efficient high-resolution diffusion models. In *Int. Conf. Learn. Represent.*, 2025c.

Junyu Chen, Dongyun Zou, Wenkun He, Junsong Chen, Enze Xie, Song Han, and Han Cai. DC-AE 1.5: Accelerating diffusion model convergence with structured latent space. *arXiv preprint arXiv:2508.00413*, 2025d.

Yu Cheng and Fajie Yuan. Leanvae: An ultra-efficient reconstruction vae for video diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15692–15702, October 2025.

Katherine Crowson, Stefan Andreas Baumann, Alex Birch, Tanishq Mathew Abraham, Daniel Z. Kaplan, and Enrico Shippole. Scalable high-resolution pixel-space image synthesis with hourglass diffusion transformers. *arXiv preprint arXiv:2401.11605*, 2024.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pp. 248–255. IEEE Computer Society, 2009. doi: 10.1109/CVPR.2009.5206848. URL `https://doi.org/10.1109/CVPR.2009.5206848`.

Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat GANs on image synthesis. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=AAWuCvzaVt.

Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pp. 12873–12883. Computer Vision Foundation / IEEE, 2021. doi: 10.1109/CVPR46437.2021.01268. URL https://openaccess.thecvf.com/content/CVPR2021/html/Esser_Taming_Transformers_for_High-Resolution_Image_Synthesis_CVPR_2021_paper.html.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Robin Rombach, and Björn Ommer. Scaling rectified flow transformers for high-resolution image synthesis. In *Int. Conf. Mach. Learn.*, 2024.

Jiayi Guo, Xingqian Xu, Yifan Pu, Zanlin Ni, Chaofei Wang, Manushree Vasu, Shiji Song, Gao Huang, and Humphrey Shi. Smooth diffusion: Crafting smooth latent spaces in diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7548–7558, 2024.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16000–16009, 2022.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Adv. Neural Inform. Process. Syst.*, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Adv. Neural Inform. Process. Syst.*, 2020.

Emiel Hoogeboom, Jonathan Heek, and Tim Salimans. Simple diffusion: End-to-end diffusion for high resolution images. In *Int. Conf. Mach. Learn.*, 2023.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Theodoros Kouzelis, Ioannis Kakogeorgiou, Spyros Gidaris, and Nikos Komodakis. EQ-VAE: Equivariance regularized latent space for improved generative image modeling. In *Int. Conf. Mach. Learn.*, 2025.

Bolin Lai et al. Toward diffusible high-dimensional latent spaces: A frequency perspective. *arXiv preprint arXiv:2511.22249*, 2025.

Xingjian Leng, Jaskirat Singh, Yunzhong Hou, Zhenchang Xing, Saining Xie, and Liang Zheng. Repa-e: Unlocking vae for end-to-end tuning with latent diffusion transformers, 2025. URL https://arxiv.org/abs/2504.10483.

Zongjian Li, Bin Lin, Yang Ye, Liuhan Chen, Xinhua Cheng, Shenghai Yuan, and Li Yuan. Wf-vae: Enhancing video vae by wavelet-driven energy flow for latent video diffusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17778–17788, June 2025.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Zhuoyan Luo, Fengyuan Shi, Yixiao Ge, Yujiu Yang, Limin Wang, and Ying Shan. Open-magvit2: An open-source project toward democratizing auto-regressive visual generation, 2025. URL https://arxiv.org/abs/2409.04410.

Nanye Ma, Mark Goldstein, Michael S Albergo, Julius Berner, and Yaron Lipman. Scalable interpolant transformers. *arXiv preprint arXiv:2401.08740*, 2024.

Diganta Misra. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*, 2019.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8162–8171. PMLR, 2021. URL http://proceedings.mlr.press/v139/nichol21a.html.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Int. Conf. Comput. Vis.*, 2023.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. SDXL: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

Konpat Preechakul, Nattanat Chatthee, Suttisak Wizadwongsa, and Supasorn Suwajanakorn. Diffusion autoencoders: Toward a meaningful and decodable representation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pp. 10609–10619. IEEE, 2022. doi: 10.1109/CVPR52688.2022.01036. URL https://doi.org/10.1109/CVPR52688.2022.01036.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Fengyuan Shi, Zhuoyan Luo, Yixiao Ge, Yujiu Yang, Ying Shan, and Limin Wang. Scalable image tokenization with index backpropagation quantization, 2025a. URL https://arxiv.org/abs/2412.02692.

Minglei Shi, Haolin Wang, Wenzhao Zheng, Ziyang Yuan, Xiaoshi Wu, Xintao Wang, Pengfei Wan, Jie Zhou, and Jiwen Lu. Latent diffusion model without variational autoencoder. *arXiv preprint arXiv:2510.15301*, 2025b.

Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.

Oriane Siméoni et al. DINOv3: Training scalable self-supervised vision foundation models. *arXiv preprint arXiv:2508.10104*, 2025.

Ivan Skorokhodov, Sharath Girish, Benran Hu, Willi Menapace, Yanyu Li, Rameen Abdal, Sergey Tulyakov, and Aliaksandr Siarohin. Improving the diffusability of autoencoders. In *Int. Conf. Mach. Learn.*, 2025.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=St1giarCHLP.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Yuchi Wang, Junliang Guo, Xinyi Xie, Tianyu He, Xu Sun, and Jiang Bian. Vidtwin: Video vae with decoupled structure and dynamics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22922–22932, June 2025.

Yuxin Wu and Kaiming He. Group normalization. In *Eur. Conf. Comput. Vis.*, 2018.

Jingfeng Yao, Cheng Wang, Wenyu Hu, Ailing Wang, and Yu Liu. Reconstruction vs. generation: Taming optimization dilemma in latent diffusion models. *arXiv preprint arXiv:2501.01423*, 2025a.

Jingfeng Yao et al. Reconstruction vs. generation: Taming optimization dilemma in latent diffusion models. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2025b.

Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation alignment for generation: Training diffusion transformers is easier than you think. In *Int. Conf. Learn. Represent.*, 2025.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Adv. Neural Inform. Process. Syst.*, 2019.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595, 2018.

Boyang Zheng, Nanye Ma, Shengbang Tong, and Saining Xie. Diffusion transformers with representation autoencoders. *arXiv preprint arXiv:2510.11690*, 2025.

A Appendix

Limitations and future work. Our modifications are encoder-only; decoder-side interactions with latent structure remain unexplored. This means our research does not accelerate diffusion inference. A preliminary combination with EQ-VAE Kouzelis et al. (2025) (Appendix E) confirms compatibility with loss-based regularisation; a thorough joint study with regularization and representation-alignment Yao et al. (2025b); Yu et al. (2025) strategies, and extension beyond ImageNet, are left for future work. In addition, we do not test the full spectrum of compute matched AE encoder architectures.

B Training Setup Details

We provide detailed hyperparameters for both the VAE training stage and the downstream diffusion training stage. All experiments use ImageNet-1k Deng et al. (2009) for both training and evaluation.

B.1 VAE Training

Our VAE training follows the setup of the original LDM Rombach et al. (2022). We use the LDM codebase with a PatchGAN Isola et al. (2017) discriminator enabled after 50,000 steps. We use an EMA of .995 and select the final checkpoint for downstream evaluation. Table 5 lists the full set of hyperparameters.

Table 5: VAE training hyperparameters.

Hyperparameter	Value
Codebase	LDM Rombach et al. (2022)
Dataset	ImageNet-1k train split
Resolution	256×256
Batch size (per GPU)	64
Gradient accumulation	1
Effective batch size	64
Optimizer	AdamW
Base learning rate	3.0×10^{-6} (linearly scaled by effective batch size)
Actual LR	1.92×10^{-4}
KL weight	1×10^{-6}
Discriminator	PatchGAN, enabled at step 50,000
PatchGAN Archicture	Same as original LDM codebase
Discriminator weight	0.5
Discriminator Optimizer/LR	Same as Base Learning Rate
EMA decay rate	0.995
Max training steps	250,000
Checkpoint selection	Final Checkpoint
GPU Used	1xB200
Memory	64GB
Time of execution	3-5 wallclock days, maximum of 250k steps
Reconstruction Loss	L2
Reconstruction weight	1.0
LPIPS weight	.5
LPIPS Network	AlexNet

B.2 Downstream Diffusion Training

For downstream evaluation we utilize the full ImageNet-1k training-set and train a class-conditional DiT-XL/2 Peebles & Xie (2023) with rectified flow Liu et al. (2022), i.e. the standard SiT Ma et al. (2024) configuration. We use a variation of LightningDiT codebase Yao et al. (2025a) as it provides the most complete and well-maintained implementation of DiT/SiT baselines. Importantly, we do *not* enable any LightningDiT-specific improvements (e.g. logit-normal time sampling, SwiGLU FFN, REPA loss); all experiments use the standard SiT configuration throughout. All gFID values are reported with class conditioning, guidance scaling is typically at (CFG scale $= 1.0$) unless reported otherwise. Table 6 lists the full set of hyperparameters.

B.3 Total compute

Including a small number of failed VAE runs, we have approximately 27 VAEs trained, each taking around 100 hours to train completely. In addition, we train 22 downstream diffusion models. We experienced two failures in training, resulting in 24 models trained total, at approximately 28-30 hours each. This results in around 3500 B200 hours used.

C Evaluation

C.1 Latent

We calculate latent normalization offline, using the entire ImageNet train set, separately for each VAE encoder. The mean is shifted to zero and the standard deviation is scaled to one, per channel. Mean/std is calculated across all images and spatial positions. Latents are calculated using posterior means to align with offline methods. Before VAE decoding (inference), we apply the inverse transformation.

We apply horizontal flips randomly when doing on-the-fly encoding.

Table 6: Downstream DiT training hyperparameters (standard SiT configuration). EMA model is used for evaluation. We use the final checkpoint.

Hyperparameter	Value
Codebase	LightningDiT Yao et al. (2025a) (standard SiT config only)
Commit Branch	f315f25b6aaad600b4d8e50a8167ce06f2e957f3
Architecture	DiT-XL/2 & DiT-B/2 Peebles & Xie (2023)
Flow matching	Rectified flow Liu et al. (2022) (linear interpolant)
Prediction target	Velocity
Time sampling	Uniform $t \sim \mathcal{U}(0, 1)$
Dataset	ImageNet-1k train split
Batch size	256
Optimiser	AdamW
Learning rate	1×10^{-4} (constant)
Weight decay	0
EMA decay	0.9999
Training steps	400,000
Conditioning	Class-conditional (adaLN-Zero, 1000 classes)
CFG scale for FID evaluation	1.0 (class-conditional, no guidance scaling)
GPU Used	1xB200
Precision	BF16
Memory	70GB
Time of execution	31 hours
Warmup	None

LSUN models use the Imagenet statistics, since that is what the VAE was trained on.

C.2 Sampling

For each VAE encoder, we compute channel-wise latent-normalization statistics offline using the entire ImageNet-1k training set. Each image is encoded using the posterior mean, $z = \mu_\phi(x)$, rather than a sample from the posterior distribution. For each latent channel c, the mean μ_c and standard deviation σ_c are computed across all training images and spatial positions. Before diffusion training, the latents are standardized as

$$\widetilde{z}c, h, w = \frac{zc, h, w - \mu_c}{\sigma_c}. \tag{9}$$

The statistics are calculated separately for every VAE encoder. Before decoding a generated latent, we apply the inverse transformation,

$$z_{c,h,w} = \widetilde{z}_{c,h,w}\sigma_c + \mu_c. \tag{10}$$

When images are encoded on the fly, we apply a random horizontal flip with probability 0.5 before VAE encoding.

For generation, we draw the initial latent from a standard Gaussian distribution, $x_0 \sim \mathcal{N}(0, I)$, and integrate the rectified-flow ODE from $t = 0$ to $t = 1$ using $K = 250$ uniform Euler updates. Specifically, with $t_i = i/K$, we apply

$$x_{i+1} = x_i + (t_{i+1} - t_i)v_\theta(x_i, t_i), \qquad i = 0, \ldots, K - 1. \tag{11}$$

No stochastic correction or additional sampling heuristic is used. Unless otherwise stated, generation is performed with classifier-free guidance scale 1.0.

We compute generation FID using 50,000 generated images and the original ADM evaluation protocol (Dhariwal & Nichol, 2021), including its Inception network, image preprocessing, and official ImageNet-256×256 reference statistics. Generated images are converted to unsigned 8-bit RGB images in the range $[0, 255]$ before evaluation. We use the same evaluation seed, 42, for every model comparison.

D LatentMaid Block Details

D.1 Encoder stages

An encoder stage tuple is $unshuffle_factor, C, M, number_of_blocks$, where C is the stage width and M is the width of each SwiGLU branch:

```
block_configs = (
    (1, 64, 128, 2),
    (2, 128, 256, 2),
    (2, 256, 512, 2),
    (2, 512, 1024, 2),
)
```

Every stage applies pixel unshuffle, then a biased 1 × 1 channel projection, then two independent BasicBlocks. Factor-1 pixel unshuffle is the identity. Stage projections have stride 1 and padding 0, with no adjacent activation or normalization. All four configured stages require a projection. There is no separate encoder stem or encoder attention.

D.2 Encoder BasicBlock and SwiGLU

The exact block order is:

```
def encoder_block(x, C, M):                    # x: [B, C, H, W]
    h = x.permute(0, 2, 3, 1)                  # [B, H, W, C]
    h = RMSNorm(C, eps=None)(h)
    h = h.permute(0, 3, 1, 2)
    h = Conv2d(C, C, 3, stride=1, padding=1,
               groups=C, bias=True)(h)
    a, b = Conv2d(C, 2*M, 1, bias=True)(h).chunk(2, dim=1)
    h = silu(a) * b
    return Conv2d(M, C, 1, bias=True)(h)
```

The depthwise convolution has a **3 × 3 kernel, stride 1, one-pixel zero padding, dilation 1, and channel multiplier 1** (groups=C). Both SwiGLU projections are biased 1 × 1 convolutions, with stride 1 and padding 0. They mix channels independently at each spatial position. The gate is SiLU(a) = a * sigmoid(a).

The branch expansion ratio is **M/C = 2**; the combined gate/value projection has **2M = 4C** channels. There is no hidden-width rounding or additional 2/3 factor. There is one normalization per block, before the depthwise convolution. There is no normalization between the convolution and SwiGLU, between the SwiGLU projections, or after the block. There is no dropout, LayerScale, or stochastic depth.

D.3 Encoder normalization and posterior

The encoder uses torch.nn.RMSNorm(C) with **eps=None**, learned per-channel gain initialized to 1, and no bias. It normalizes channels only, separately at every batch and spatial position:

$$y[b,c,h,w] = gamma[c] * x[b,c,h,w]/sqrt(mean_over_c(x[b,c,h,w]^2) + epsilon) \quad (12)$$

There is no mean subtraction, spatial reduction, batch reduction, or running statistic. The numeric epsilon is supplied by the installed PyTorch version; the experiment source does not fix it to 1e-6. Record the PyTorch version and dtype when reproducing this behavior.

After the final encoder stage:

```
h = Conv2d(512, 8, 1, bias=True)(h)            # encoder.out_proj
moments = Conv2d(8, 8, 1, bias=True)(h)        # AutoencoderKL.quant_conv
```

```
mean, logvar = moments.chunk(2, dim=1)
logvar = logvar.clamp(-30.0, 20.0)
z = mean + exp(0.5 * logvar) * noise
```

Both projections have stride 1 and padding 0, with no activation or normalization before, between, or after them. noise is standard normal. DiagonalGaussianDistribution performs the split and clamp; its mode() returns the mean for deterministic decoding. The first four channels after quant_conv are mean and the last four are log variance. There is no latent normalization in this autoencoder path.

D.4 Initialization

These are fresh-construction defaults; loading a checkpoint replaces them. All convolution weights except encoder output projection retain the PyTorch Conv2d default Kaiming Uniform, equivalent to uniform bounds Their biases use the same uniform bounds. This includes stage projections, depthwise filters, SwiGLU projections, both quant projections, and every decoder convolution, including its final RGB head.

The encoder explicitly initializes both weight and bias of the output projection from a zero-mean normal distribution with standard deviation **1/8**. The constructor doubles its default latent width from 4 to 8 before computing std=1.0/latent_dim. It does not zero any log-variance channels. A subsequent randomly initialized quant_conv mixes the eight output channels before the posterior split, so initial log variance is not constrained to zero.

RMSNorm gains start at 1. GroupNorm gains start at 1 and biases at 0. There is no special zero initialization of decoder residual branches or attention output projections.

E Additional Quantitative Results

This appendix collects supplementary quantitative results that lie outside the default $f8c4$ / CFG=1 evaluation regime reported in the main text. We provide (i) a classifier-free guidance (CFG) sweep for the headline configurations at $f8c4$, with and without EQ-VAE regularisation Kouzelis et al. (2025) (Sec. E.1); (ii) results in a more aggressive $f16c16$ compression regime (Sec. E.2); (iii) a brief synthesis of the EQ-VAE combination across compression regimes (Sec. E.3); and (iv) per-step convergence checkpoints for the headline configurations (Sec. E.5). All values are class-conditional gFID on the ImageNet-1k validation set (50,000 images), under the same DiT-XL/2 backbone trained for 400k steps unless stated otherwise.

E.1 Classifier-Free Guidance Sweep at $f8c4$

Tab. 7 sweeps CFG for the four headline encoders at the default $f8c4$ compression, with and without EQ-VAE regularisation. Two patterns emerge. First, the CFG=1 ranking does not persist as guidance is increased: at the standard operating point CFG$\in[1.75, 2.0]$, all four unregularised encoders converge into a narrow gFID band (4.5–5.1), indicating that the CFG=1 gFID gap reported in the main text is not a guidance-specific artifact and that locality-preserving variants do not lose ground when guidance is applied - methods tend to converge under guidance. Second, with EQ-VAE regularization the best gFID drops from 4.74 to 4.33 (CFG=1.75) for the ABCRP standard block and from 4.87 to 4.76 (CFG=2.0) for the LatentMaid block, suggesting that architectural locality and equivariance regularisation are composable.

E.2 Higher-Compression Regime ($f16c16$)

We additionally evaluate the headline $2/2/2/2$ configurations under a more aggressive compression: $16\times$ spatial reduction with 16-channel latents ($f16c16$), paired with DiT-XL using patch size 2. This setting reduces the effective number of latent tokens substantially compared with the default $f8c4$ configuration, and reflects the regime studied by DC-AE Chen et al. (2025c;d).

Plain $f16c16$ latents are noticeably harder for diffusion training: at cfg=1, gFID rises to 24.94 for ABCRP+std. and 23.30 for LatentMaid (Tab. 8). Adding EQ-VAE regularisation closes most of the gap, recovering cfg=1 gFID to 17.76 and 18.91, respectively. LatentMaid + EQ remains

Table 7: gFID across CFG values for the headline configurations at $f8c4$, with and without EQ-VAE regularisation. All encoders share the same DiT-XL/2 backbone trained for 400k steps; gFID is reported on the full $50{,}000$-image ImageNet-1k validation set.

Config	CFG=1.0	1.25	1.5	1.75	2.0
A (SD-VAE baseline)	19.89	11.42	6.88	5.09	4.80
ABCRP + std. block, $2/2/2/2$	17.68	10.60	6.90	5.13	4.74
ABCRP + LatentMaid, $2/2/2/2$	18.32	10.53	6.73	4.98	4.87
ABCRP + std. block, $2/2/2/2$ + EQ-VAE	16.08	8.37	5.07	4.33	4.90
ABCRP + LatentMaid, $2/2/2/2$ + EQ-VAE	18.32	10.55	6.59	5.02	4.76

within roughly one gFID point of the standard ABCRP + EQ encoder across the full CFG sweep, indicating that locality-preserving design transfers to higher compression ratios when paired with a complementary regulariser.

Table 8: gFID at $f16c16$ (DiT patch size 2) across CFG values, with and without EQ-VAE Kouzelis et al. (2025) regularisation. All values use the $2/2/2/2$ stage allocation and share the same DiT-XL backbone.

Config	CFG=1.0	1.25	1.5	1.75	2.0	2.25
ABCRP + std. block	24.94	22.63	20.49	18.58	16.87	15.30
+ EQ-VAE	17.76	15.81	14.07	12.55	11.20	10.00
ABCRP + LatentMaid	23.30	20.32	17.77	15.55	13.64	12.04
+ EQ-VAE	18.91	16.54	14.53	12.80	11.33	10.06

To match the latent token count of the default $f8c4$ configuration (256 tokens), we additionally evaluate $f16c16$ at DiT patch size 1 with EQ-VAE regularisation (Tab. 9). At CFG=1, ABCRP+std. reaches gFID 13.79 and ABCRP+LatentMaid reaches 15.57, both substantially better than their patch-2 counterparts (17.76 and 18.91): the larger token budget recovers most of the compression-induced cost.

Table 9: gFID at $f16c16$ with DiT patch size 1 (matching the 256-token budget of the default $f8c4$ configuration), with EQ-VAE Kouzelis et al. (2025) regularisation. All values use the $2/2/2/2$ stage allocation.

Config	CFG=1.0	1.25	1.5	1.75	2.0
ABCRP + std. block + EQ-VAE	13.79	12.01	10.48	9.17	8.06
ABCRP + LatentMaid + EQ-VAE	15.57	13.38	11.54	9.99	8.68

Pushing CFG further into the high-guidance regime, both encoders continue to improve and bottom out around CFG$\in[4.0, 4.5]$ (Tab. 10). The best gFID under EQ-VAE is 3.60 for ABCRP+std. (CFG=4.5) and 4.00 for ABCRP+LatentMaid (CFG=4.5), competitive with the best $f8c4$ values reported earlier.

E.3 Compatibility with EQ-VAE Regularisation

Our locality-preserving design is complementary to loss-based regularisation rather than a substitute for it. At the default $f8c4$ compression (Tab. 7), EQ-VAE on top of ABCRP+std. improves CFG=1 gFID from 17.68 to 16.08 and the optimal-CFG gFID from 4.74 to 4.33; on LatentMaid the CFG=1 value is essentially unchanged (18.32) but the optimal-CFG gFID improves from 4.87 to 4.76. The benefit is much larger in the higher-compression regime (Tab. 8), where regularisation evidently helps to compensate for the reduced token budget. Together, these results suggest that architectural locality and equivariance regularisation address overlapping but non-identical sources of latent-side instability; a more thorough joint study is left to future work.

Table 10: High-CFG sweep at $f16c16$ with DiT patch size 1 and EQ-VAE Kouzelis et al. (2025) regularisation, $2/2/2/2$ stage allocation.

Config	CFG=2.5	3.0	3.5	4.0	4.5
ABCRP + std. block + EQ-VAE	6.32	5.11	4.32	3.84	3.60
ABCRP + LatentMaid + EQ-VAE	6.75	5.43	4.60	4.15	4.00

E.4 rFID vs gFID Across Configurations

We collect the rFID/gFID pairs from every configuration trained in this paper (Tab. 11) to support the observation that, even within a single architectural family on a single dataset, reconstruction fidelity is a poor predictor of downstream generation quality. Across our $N=12$ encoder configurations (all with the same decoder, training schedule, and DiT backbone), Pearson $r=0.382$ and Spearman $\rho=0.534$. Concretely, the configuration with the lowest rFID (**0.63**, ABCRP+std. at $2/2/2/4$) ranks only seventh in gFID (19.17), while the best gFID (**17.68**, ABCRP+std. at $2/2/2/2$) has a notably higher rFID (0.70).

Table 11: rFID and gFID for every configuration trained in this paper, sorted by rFID. Same DiT-XL/2 backbone (400k steps, CFG=1) and ImageNet-1k validation set throughout. Within this set Pearson $r(\text{rFID}, \text{gFID})=0.382$, Spearman $\rho=0.534$.

Block family	Stage config	rFID ↓	gFID ↓
ABCRP + std. block	$2/2/2/4$	0.63	19.17
ABCRP + std. block	$2/2/2/2$	0.70	17.68
ABCRP + std. block	$0/2/2/2$	0.72	18.27
ABCRP + std. block	$0/0/2/2$	0.81	19.16
ABCRP + LatentMaid	$0/2/2/2$	0.82	19.29
ABCRP + std. block	$0/0/0/2$	0.85	18.40
ABCRP + LatentMaid	$2/2/2/2$	0.85	18.32
ABCRP + LatentMaid	$0/0/2/2$	0.88	20.46
ABCRP + LatentMaid	$2/2/2/4$	0.92	23.07
AB (no enc. attn)	$2/2/2/4$	0.95	17.92
A (SD-VAE baseline)	$2/2/2/4$	1.39	19.89
ABCRP + LatentMaid	$0/0/0/2$	1.40	22.58

E.5 Per-Step Convergence Checkpoints

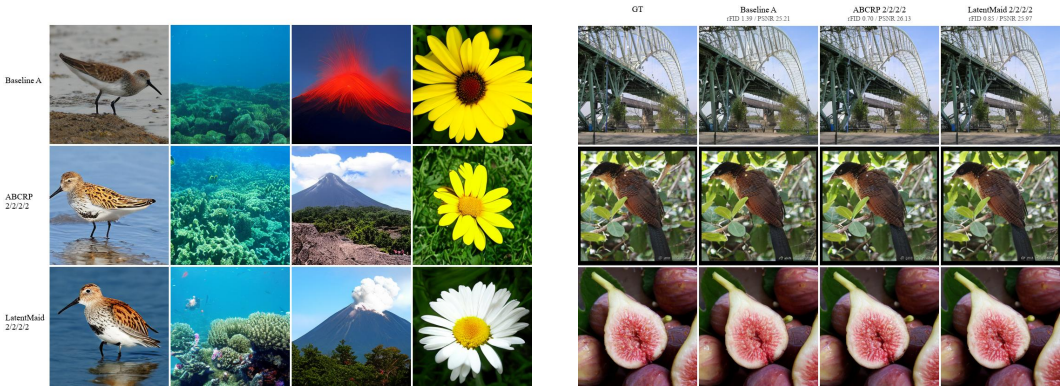
Tab. 12 reports the CFG=1 gFID checkpoints used to construct Fig. 1. Linearly interpolating between the 300k and 400k checkpoints, ABCRP+std. at $2/2/2/2$ matches the SD-VAE baseline's 400k gFID (19.89) at approximately 325k DiT steps, and ABCRP+LatentMaid at $2/2/2/2$ matches it at approximately 346k steps, corresponding to the $1.23\times$ and $1.16\times$ training-step speedups reported in the main text.

Table 12: Downstream CFG=1 gFID at intermediate DiT-XL/2 training steps for the SD-VAE baseline, ABCRP+std., and ABCRP+LatentMaid (both at $2/2/2/2$). The 400k column reproduces the headline numbers from Tab. 4.

Config	100k	150k	200k	250k	300k	400k
A (SD-VAE baseline)	43.96	33.98	29.07	25.72	23.37	19.89
ABCRP + std. block, $2/2/2/2$	41.08	31.02	25.97	22.75	20.64	17.68
ABCRP + LatentMaid, $2/2/2/2$	41.52	32.01	26.57	23.42	21.24	18.32

Table 13: Cumulative component ablation at fixed stage allocation $2/2/2/4$. Each row adds one modification to the previous configuration. Beginning with AB, attention is removed from both the encoder and decoder. Model is DiT-XL/2 for 400K steps at CFG=1

Config	Modification	rFID ↓	PSNR ↑	gFID ↓
A	Baseline	1.39	25.21	19.89
AB	− attention	0.862	25.65	22.96
ABC	+ Mish	0.915	23.58	21.93
ABCR	+ RMSNorm	0.909	25.39	21.57
ABCRP	+ pixel unshuffle	0.887	25.71	22.6



(a) Generation samples (CFG=4, 50 steps). (b) Reconstruction.

Figure 4: Qualitative comparison. (a) DiT-XL/2 generation samples, (b) VAE reconstruction.

F Addtional Qualitative Results

Fig. 4 compares the SD-VAE baseline **A** against the two best $2/2/2/2$ encoders (ABCRP+std. block and ABCRP+LatentMaid) on both ends of the pipeline. For generation, samples drawn from the DiT-XL/2 trained on each encoder's latents (CFG=4, 50 steps) show that the locality-preserving encoders produce class-conditional samples with cleaner global structure (sandpiper anatomy, volcano silhouette) and sharper local detail (coral texture, daisy petals) than the SD-VAE baseline. For reconstruction, both ABCRP variants closely match the ground truth on fine high-frequency content (bridge truss, plumage, fig flesh) consistent with their lower rFID.

G Related Work Continued

Compression-driven approaches such as DC-AE Chen et al. (2025c) push spatial compression to f_{32}–f_{128} for faster diffusion training, while DC-AE 1.5 Chen et al. (2025d) shows that increasing latent channel capacity improves reconstruction (rFID) yet *degrades* generation (gFID) due to sparse object structure in high-dimensional latents Luo et al. (2025); Shi et al. (2025a). *Representation-alignment* strategies align latents with pretrained vision foundation models (VFMs): VA-VAE Yao et al. (2025b) aligns VAE latents with DINOv2 Oquab et al. (2023), MAETok Chen et al. (2025b) replaces the encoder with a masked autoencoder He et al. (2022), REPA Yu et al. (2025); Leng et al. (2025) regularizes diffusion hidden states toward VFM features, and RAE Zheng et al. (2025) and SVG Shi et al. (2025b) bypass the VAE encoder.

G.1 Encoder Receptive Field & Global Operations

Tab. 15 demonstrates the reduction in both receptive field and global operations associated with each set of changes.

G.2 Locality & Smoothness

G.2.1 Protocols

We evaluate nine of the main models - the full ablation chain from A to ABCRP, with and without decoder attention. Evaluation set consisted of images from the Imagenet validation set, with no duplicates.

G.2.2 Evaluated Metrics

Neighbor cosine similarity: do neighboring locations represent information in similar directions? Higher means that adjacent latents are more aligned, and thus smoother. The centered version removed constant channel offsets to get more accurate similarity. 16 Normalized Total Variation: How abruptly does the representation change between neighboring locations? Lower means the spatial changes are smooth, and normalization takes into account differences in latent scale.

We see that ABCRP, with and without encoder attention, is moderately more smooth. This is highlighted in the centered neighbor cosine, and while present in TV Normalized, does not appear especially significant.

17 HF Energy Fraction: How much of the representation is just rapid spatial fluctuations? Lower means lower fine-scale variation - but this cannot distinguish between latent noise and latent details. Nonetheless, we see that ABCRP with and without decoder attention are the lowest.

18 Translation Error: When the image moves, does the latent move with it while retaining the same values? Lower is better. We find that ABCR (without decoder attention) wins four out of the eight, while ABCRP *with* decoder attention wins three out of the eight.

21 Dense Metrics:

Correspondence PCK: What fraction of pixel locations are matched within one latent cell of their correct position. This is a synthetic task for testing dense downstream tasks with the VAE. This is accuracy, so higher is better. ABCRP is the best option for right and left - ABCRP with decoder attention slightly outperforms for down and up.

Correspondence Endpoint Error: When matching a latent to the shifted image, how far away is the selected match from the correct position? Lower means more accurate spatial matching. ABCRP distinctly outperforms all other models in this testing.

Effective receptive-field radii: How widely distributed are the input pixels that influence the central latent token? Smaller means more localized - not better or worse, just a localization test. We find, that as intended by our model construction, ABCRP has the smallest effective radius. 19

Jacobian Energy: How strongly does the central latent respond to small changes in the input pixels? Higher means greater local sensitivity. As intended with our architecture changes, ABCRP has the least sensitivity to small changes in input changes. 20

G.2.3 Discussion

Across several complementary diagnostics, the efficient encoder retains well-behaved spatial representations and shows modest improvements in smoothness, correspondence, and sensitivity localization. These changes are not monotonic across every component or metric, suggesting that no individual locality statistic explains downstream performance.

Representation diagnostics. Our primary objective is not to maximize any individual latent-locality metric, but to reduce encoder and downstream training compute without compromising generation quality. We nevertheless evaluate several properties of the resulting representations to check whether the architectural simplifications introduce undesirable spatial behavior. Relative to A, ABCRP produces modestly smoother latents: normalized total variation decreases from 1.049 to 1.002, centered neighboring-token cosine similarity increases from 0.084 to 0.115, and high-frequency energy decreases from 75.3% to 73.7%. Spatial correspondence also improves, with rightward matching error decreasing from 0.315 to 0.125 latent cells and exact accuracy increasing from 98.02% to 99.12%. The 90% Jacobian-energy radius decreases from 15.69 to 12.53 input pixels, although this estimate is based on eight images and should be interpreted as preliminary.

These trends are not uniform. ABCRP slightly increases rightward normalized translation error, from 0.0325 to 0.0336, and ABCR achieves lower translation error than ABCRP in all four directions. Reconstruction PSNR also decreases slightly, from 25.32 to 25.18 dB. We therefore interpret these measurements as descriptive diagnostics rather than evidence that a single notion of locality causally determines downstream performance. The central result is instead the compute–quality trade-off: ABCRP substantially reduces encoder cost while preserving reconstruction and improving or maintaining downstream generation quality.

H FLOPs Calculation

In this section, we discuss the FLOPs associated with training various models. We assume a batch size of 64. The following table is per-step.

Additional sources of VAE FLOPs are: sampling the latent, discriminator, LPIPS, loss, adaptive weight, optimizers, gradient clipping, gradient accumulation calculations, EMA, KL. KL is counted under loss.

Various components are fixed cost. They are listed in table 23. We do add them in to: training forward, training total, Params (Enc + Dec).

For *all* counts, we consider MAC as two FLOPs. See 23

H.1 Percentage Claims

LatentMaid encoder forward costs are 1.90 TFLOPS. Config A is 17.51. This is around %10.85, giving us 89.15% reduction in encoder forward costs.

The 35% reduction results from both LatentMaid's lower per-step encoder cost and its faster downstream convergence. We use Config A's final gFID of 19.89 at 400k steps as the target and linearly interpolate between LatentMaid's 300k and 400k checkpoints, finding that LatentMaid reaches the same gFID at approximately 346k steps. Comparing the cumulative DiT forward/backward and online encoder-forward FLOPs at these matched-quality points yields a 35% reduction. This calculation excludes VAE pretraining and assumes that latents are encoded online rather than cached.

Including VAE pretraining and one-time latent caching, LatentMaid matches the baseline's 400k-step gFID with approximately 19% fewer total training FLOPs.

I LatentMaid Block Details

I.1 Encoder stages

An encoder stage tuple is $unshuffle_factor, C, M, number_of_blocks$, where C is the stage width and M is the width of each SwiGLU branch:

```
block_configs = (
    (1, 64, 128, 2),
    (2, 128, 256, 2),
    (2, 256, 512, 2),
    (2, 512, 1024, 2),
)
```

Every stage applies pixel unshuffle, then a biased 1 × 1 channel projection, then two independent BasicBlocks. Factor-1 pixel unshuffle is the identity. Stage projections have stride 1 and padding 0, with no adjacent activation or normalization. All four configured stages require a projection. There is no separate encoder stem or encoder attention.

I.2 Encoder BasicBlock and SwiGLU

The exact block order is:

```
def encoder_block(x, C, M):                    # x: [B, C, H, W]
    h = x.permute(0, 2, 3, 1)                  # [B, H, W, C]
    h = RMSNorm(C, eps=None)(h)
    h = h.permute(0, 3, 1, 2)
    h = Conv2d(C, C, 3, stride=1, padding=1,
               groups=C, bias=True)(h)
    a, b = Conv2d(C, 2*M, 1, bias=True)(h).chunk(2, dim=1)
    h = silu(a) * b
    return Conv2d(M, C, 1, bias=True)(h)
```

The depthwise convolution has a **3 × 3 kernel, stride 1, one-pixel zero padding, dilation 1, and channel multiplier 1** (groups=C). Both SwiGLU projections are biased 1 × 1 convolutions, with stride 1 and padding 0. They mix channels independently at each spatial position. The gate is SiLU(a) = a * sigmoid(a).

The branch expansion ratio is **M/C = 2**; the combined gate/value projection has **2M = 4C** channels. There is no hidden-width rounding or additional 2/3 factor. There is one normalization per block, before the depthwise convolution. There is no normalization between the convolution and SwiGLU, between the SwiGLU projections, or after the block. There is no dropout, LayerScale, or stochastic depth.

I.3 Encoder normalization and posterior

The encoder uses torch.nn.RMSNorm(C) with **eps=None**, learned per-channel gain initialized to 1, and no bias. It normalizes channels only, separately at every batch and spatial position:

$$y[b,c,h,w] = gamma[c] * x[b,c,h,w]/sqrt(mean_over_c(x[b,c,h,w]^2) + epsilon) \quad (13)$$

There is no mean subtraction, spatial reduction, batch reduction, or running statistic. The numeric epsilon is supplied by the installed PyTorch version; the experiment source does not fix it to 1e-6. Record the PyTorch version and dtype when reproducing this behavior.

After the final encoder stage:

```
h = Conv2d(512, 8, 1, bias=True)(h)            # encoder.out_proj
moments = Conv2d(8, 8, 1, bias=True)(h)        # AutoencoderKL.quant_conv
mean, logvar = moments.chunk(2, dim=1)
logvar = logvar.clamp(-30.0, 20.0)
z = mean + exp(0.5 * logvar) * noise
```

Both projections have stride 1 and padding 0, with no activation or normalization before, between, or after them. noise is standard normal. DiagonalGaussianDistribution performs the split and clamp; its mode() returns the mean for deterministic decoding. The first four channels after quant_conv are mean and the last four are log variance. There is no latent normalization in this autoencoder path.

I.4 Initialization

These are fresh-construction defaults; loading a checkpoint replaces them. All convolution weights except encoder output projection retain the PyTorch Conv2d default Kaiming Uniform, equivalent to uniform bounds Their biases use the same uniform bounds. This includes stage projections, depthwise filters, SwiGLU projections, both quant projections, and every decoder convolution, including its final RGB head.

The encoder explicitly initializes both weight and bias of the output projection from a zero-mean normal distribution with standard deviation **1/8**. The constructor doubles its default latent width from 4 to 8 before computing std=1.0/latent_dim. It does not zero any log-variance channels. A subsequent randomly initialized quant_conv mixes the eight output channels before the posterior split, so initial log variance is not constrained to zero.

RMSNorm gains start at 1. GroupNorm gains start at 1 and biases at 0. There is no special zero initialization of decoder residual branches or attention output projections.

Table 14: **Scope of comparison with related autoencoder directions.** Our controlled comparison concerns encoder-side efficiency under a fixed latent interface ($f8c4$), decoder, VAE objective, and downstream diffusion protocol. Methods targeting other axes are complementary, but are not matched encoder-efficiency baselines. We therefore evaluate EQ-VAE and the LightningDIT training stack through composition with our encoder, while comparing directly against reduced standard LDM encoders.

Direction	**Primary optimization axis**	**Changes relative to our controlled interface**	**Matched encoder-efficiency baseline?**	**Role in our evaluation**
Compression-oriented autoencoders (e.g., DC-AE/DC-AE 1.5)	Latent token count and downstream diffusion cost	Typically changes the spatial compression ratio, latent dimensionality, or complete autoencoding system	No	Studied as a complementary compression direction; we separately test whether our design transfers to the higher-compression $f16c16$ regime.
EQ-VAE	Spatial equivariance and latent regularization	Modifies the VAE training objective rather than primarily reducing encoder architecture cost	No	Combined directly with our encoder to test compatibility between architectural efficiency and loss-based regularization.
Alternative tokenizer paradigms (e.g., MAETok, RAE, SVG)	Latent semantics or tokenizer formulation	Replaces or substantially changes the encoder, latent representation, training objective, or downstream interface	No	Included as related but non-controlled directions; direct numerical comparison would conflate encoder efficiency with changes to the representation and training protocol.
Reduced standard LDM encoders	Encoder parameters and computation	Preserves the latent interface, decoder, objective, and downstream protocol	Yes	Used as matched efficiency controls through attention removal, reduced stage allocations, and standard residual blocks.
LatentMaid (ours)	**Encoder architecture and compute allocation**	**Preserves the latent interface, decoder, objective, and downstream protocol**	**Yes**	**Evaluated against matched standard-encoder controls and in combination with EQ-VAE and the VA-VAE training stack.**

Table 15: Receptive field characteristics of the different architectures.

Architecture	Level	Stride	Local RF	Full RF	Global sources
A	Input	1	1×1	1×1	–
	Input conv	1	3×3	3×3	–
	Stage 1 (2 blocks)	1	11×11	Global	GroupNorm
	Downsample 1	2	13×13	Global	GroupNorm
	Stage 2 (2 blocks)	2	29×29	Global	GroupNorm
	Downsample 2	4	33×33	Global	GroupNorm
	Stage 3 (2 blocks)	4	65×65	Global	GroupNorm
	Downsample 3	8	73×73	Global	GroupNorm
	Stage 4 (4 blocks)	8	201×201	Global	GroupNorm, attention
	Posterior head	8	217×217	Global	GroupNorm, attention
AB	Input	1	1×1	1×1	–
	Input conv	1	3×3	3×3	–
	Stage 1 (2 blocks)	1	11×11	Global	GroupNorm
	Downsample 1	2	13×13	Global	GroupNorm
	Stage 2 (2 blocks)	2	29×29	Global	GroupNorm
	Downsample 2	4	33×33	Global	GroupNorm
	Stage 3 (2 blocks)	4	65×65	Global	GroupNorm
	Downsample 3	8	73×73	Global	GroupNorm
	Stage 4 (4 blocks)	8	201×201	Global	GroupNorm
	Posterior head	8	217×217	Global	GroupNorm
ABC	Input	1	1×1	1×1	–
	Input conv	1	3×3	3×3	–
	Stage 1 (2 blocks)	1	11×11	Global	GroupNorm
	Downsample 1	2	13×13	Global	GroupNorm
	Stage 2 (2 blocks)	2	29×29	Global	GroupNorm
	Downsample 2	4	33×33	Global	GroupNorm
	Stage 3 (2 blocks)	4	65×65	Global	GroupNorm
	Downsample 3	8	73×73	Global	GroupNorm
	Stage 4 (4 blocks)	8	201×201	Global	GroupNorm
	Posterior head	8	217×217	Global	GroupNorm
ABCR	Input	1	1×1	1×1	–
	Input conv	1	3×3	3×3	–
	Stage 1 (2 blocks)	1	11×11	11×11	–
	Downsample 1	2	13×13	13×13	–
	Stage 2 (2 blocks)	2	29×29	29×29	–
	Downsample 2	4	33×33	33×33	–
	Stage 3 (2 blocks)	4	65×65	65×65	–
	Downsample 3	8	73×73	73×73	–
	Stage 4 (4 blocks)	8	201×201	201×201	–
	Posterior head	8	217×217	217×217	–
ABCRP	Input	1	1×1	1×1	–
	Unshuffle 1 + mix	2	2×2	2×2	–
	Stage 1 (2 blocks)	2	10×10	10×10	–
	Unshuffle 2 + mix	4	12×12	12×12	–
	Stage 2 (2 blocks)	4	28×28	28×28	–
	Unshuffle 3 + mix	8	32×32	32×32	–
	Stage 3 (2 blocks)	8	64×64	64×64	–
	Stage 4 (4 blocks)	8	128×128	128×128	–
	Posterior head	8	128×128	128×128	–

Configuration	Centered Neighbor Cosine ↑	TV Normalized ↓
Config A	0.084108	1.048509
Config AB	0.080145	1.054082
Config ABC	0.076229	1.050843
Config ABCR	0.100198	1.039518
Config ABCRP	**0.115463**	**1.002417**
Config ABCRP with Decoder Attention	0.110093	1.014197
Config ABCR with Decoder Attention	0.101413	1.042454
Config ABC with Decoder Attention	0.084207	1.045586
Config AB with Decoder Attention	0.086124	1.047900

Table 16: Evaluation metrics across model configurations.

Model	hf_energy_fraction↓
Config A	0.752785
Config AB	0.759188
Config ABC	0.760855
Config ABCR	0.744294
Config ABCRP	0.737429
Config ABCRP with Decoder Attention	**0.736660**
Config ABCR with Decoder Attention	0.745557
Config ABC with Decoder Attention	0.753396
Config AB with Decoder Attention	0.752297

Table 17: HF energy fraction for each model configuration.

Table 18: Translation error metrics for different model configurations. Lower values indicate lower error.

Model	Left		Right		Up		Down	
	RMSE↓	NRMSE↓	RMSE↓	NRMSE↓	RMSE↓	NRMSE ↓	RMSE↓	NRMSE↓
Config A	0.03169	0.03249	0.03169	0.03245	0.03140	0.03210	0.03210	0.03289
Config AB	0.03431	0.03508	0.03632	0.03723	0.02918	0.02988	0.02913	0.02983
Config ABC	0.03222	0.03308	0.03286	0.03373	0.03231	0.03313	0.03345	0.03434
Config ABCR	**0.02887**	**0.02879**	0.02942	**0.02939**	**0.02816**	0.02821	0.02838	0.02829
Config ABCRP	0.03345	0.02995	0.03730	0.03356	0.03368	0.03020	0.03356	0.02989
Config ABCRP with Decoder Attention	0.04333	0.03952	0.04133	0.03733	0.02865	**0.02572**	**0.02755**	**0.02493**
Config ABCR with Decoder Attention	0.03631	0.03599	0.03780	0.03764	0.02952	0.02925	0.03089	0.03070
Config ABC with Decoder Attention	0.03069	0.03126	0.03072	0.03132	0.03062	0.03124	0.03031	0.03084
Config AB with Decoder Attention	0.02934	0.03006	**0.02891**	0.02973	0.02896	0.02978	0.03010	0.03090

Model	ERF Radius 50	ERF Radius 90
Config A	7.382	15.686
Config AB	6.626	15.414
Config ABC	7.829	14.825
Config ABCR	6.626	14.944
Config ABCRP	**5.306**	**12.528**
Config ABCRP with Decoder Attention	6.240	14.065
Config ABCR with Decoder Attention	7.691	16.226
Config ABC with Decoder Attention	7.560	15.881
Config AB with Decoder Attention	6.804	14.856

Table 19: Effective receptive field (ERF) measurements across model configurations.

Model	Jacobian Energy
Config A	349.2922
Config AB	689.8307
Config ABC	1088.1389
Config ABCR	16052.1487
Config ABCRP	**52.3615**
Config ABCRP with Decoder Attention	171.8415
Config ABCR with Decoder Attention	355.1272
Config ABC with Decoder Attention	508.9556
Config AB with Decoder Attention	897.4708

Table 20: Jacobian energy across model configurations.

Table 21: PCK at 1-cell matching accuracy across model configurations.

Model	Right	Left	Down	Up
Config A	0.980717	0.980701	0.980515	0.981456
Config AB	0.976159	0.976903	0.981683	0.982644
Config ABC	0.978536	0.978831	0.977942	0.979673
Config ABCR	0.983956	0.984416	0.984607	0.984483
Config ABCRP	**0.991831**	**0.992332**	0.992472	0.992410
Config ABCRP with Decoder Attention	0.989888	0.989609	**0.993397**	**0.993448**
Config ABCR with Decoder Attention	0.976526	0.977741	0.983378	0.983171
Config ABC with Decoder Attention	0.980469	0.981022	0.981533	0.981719
Config AB with Decoder Attention	0.982479	0.982685	0.981926	0.982747

Model	Right Match EPE	Left Match EPE	Down Match EPE	Up Match EPE
Config A	0.3152	0.3116	0.3043	0.2921
Config AB	0.3885	0.3734	0.2800	0.2680
Config ABC	0.3427	0.3440	0.3372	0.3188
Config ABCR	0.2614	0.2551	0.2454	0.2470
Config ABCRP	**0.1253**	**0.1148**	**0.1064**	**0.1093**
Config ABCRP with Decoder Attention	0.1552	0.1571	0.0920	0.0895
Config ABCR with Decoder Attention	0.3901	0.3679	0.2718	0.2734
Config ABC with Decoder Attention	0.3142	0.3087	0.2886	0.2914
Config AB with Decoder Attention	0.2840	0.2774	0.2782	0.2686

Table 22: End-point error (EPE) in cells for matching in the right, left, down, and up directions.

Metric
Decoder Forward: 39.92 TFLOPs
Decoder Backward: 79.77 TFLOPs
Decoder Parameters: 49.49 M
Sampling Forward: 0.000001 TFLOPs
Sampling Backward: 0.0000008 TFLOPs
Discriminator Forward: 1.2112 TFLOPs
Discriminator Backward: 2.0046 TFLOPs
Discriminator Parameters: 2.77 M
LPIPS Forward: 5.1430 TFLOPs
LPIPS Backward: 2.5751 TFLOPs
LPIPS Parameters: 14.72 M
Loss Forward: 0.000103 TFLOPs
Loss Backward: 0.000065 TFLOPs
Adaptive Weight: 0.0580 TFLOPs
Generator Optimizer: 0.0012 TFLOPs
Discriminator Optimizer: 0.000039 TFLOPs
Gradient Clipping: 0.000266 TFLOPs
Gradient Accumulation: 0.000003 TFLOPs
EMA: 0.000258 TFLOPs

Table 23: Computational cost and parameter counts.

Table 24: Computational cost and parameter count at BS 64. All variants have decoder attention. FLOPs are reported in TFLOPs and parameters in millions (M). Model fwd/bwd include encoder/decoder only. Training fwd and Training total include the fixed costs.

Config	Enc Fwd	Model Fwd	Enc Bwd	Model Bwd	Combined	Enc Params	Params (Enc + Dec) only	Training Fwd	Training Total
`Config A`	17.51	57.43	34.95	114.72	172.15	34.16M	83.65M	63.78	183.14
`Config AB`	17.23	57.15	34.40	114.17	171.32	33.11M	82.60M	63.51	182.32
`Config ABC`	17.23	57.15	34.42	114.19	171.34	33.11M	82.60M	63.51	182.34
`Config ABCR`	17.22	57.14	34.41	114.18	171.32	33.10M	82.59M	63.50	182.31
`Config ABCRP`	16.71	56.63	33.38	113.15	169.78	31.38M	80.87M	62.98	180.77
`Config ABCRP 2222`	15.47	55.39	30.90	110.67	166.06	21.94M	71.43M	61.74	177.05
`LatentMaid 2222`	1.90	41.82	3.78	83.55	125.37	4.90M	54.39M	48.17	136.36

Table 25: Measured CUDA VAE inference performance at BS 64 on NVIDIA B200. All measurements use bf16 and compiled execution. E2E includes encoder, sampling, and decoder. Throughput is reported in images/s, latency in ms/batch, and peak memory in GiB. Warmup is excluded; measurements use synchronized wall time with GPU-resident inputs, random weights, and no I/O or training. Encoder measurements exclude the decoder. This does *not* affect diffusion inference speed due to the unchanged decoder.

Config	Enc Img/s	Enc ms	Enc Alloc	Enc Reserved	E2E Img/s	E2E ms	E2E Alloc	E2E Reserved
`Config A`	615.40	13.000	0.884	1.041	238.54	33.537	1.509	1.742
`Config AB`	626.39	12.772	0.880	1.041	242.03	33.054	1.504	1.740
`Config AB Dec Att`	626.80	12.763	0.880	1.041	240.25	33.298	1.507	1.740
`Config ABC`	587.23	13.623	0.880	1.041	235.91	33.911	1.504	1.740
`Config ABC Dec Att`	587.11	13.626	0.880	1.041	234.15	34.166	1.507	1.740
`Config ABCR`	1094.33	7.310	0.882	1.041	289.92	27.594	1.504	1.740
`Config ABCR Dec Att`	1094.23	7.311	0.882	1.041	287.31	27.844	1.507	1.742
`Config ABCRP`	1808.24	4.424	0.623	0.643	323.59	24.722	1.495	1.842
`Config ABCRP Dec Att`	1809.67	4.421	0.623	0.643	320.32	24.975	1.499	1.844
`Config ABCRP Dec Att 2222`	1915.06	4.177	0.588	0.604	323.79	24.707	1.464	1.805
`LatentMaid 2222`	2122.65	3.769	0.555	0.795	328.60	24.346	1.433	1.711

Table 26: Computational cost at batch size 1 with a f8c4 latent. Parameters are in millions. Forward, backward, and per-step costs are reported in TFLOPs; cumulative 400K-step costs are reported in EFLOPs.

Preset	Input Shape	Parameters	Forward	Backward	Combined FLOPs	400K-step Cost (256 BS)
DiT-B/2	$1 \times 4 \times 32 \times 32$	130.304M	0.046	0.092	0.138	**14.13**
DiT-XL/2	$1 \times 4 \times 32 \times 32$	674.816M	0.237	0.475	0.712	**72.91**

Table 27: Computation Costs for VAE, EFLOPs

VAE Configuration	Train VAE, 250K steps	Encode cached dataset twice	Encode BS 256 for 400K steps
Config A	45.79	0.70	28.02
Config ABCRP	45.19	0.67	26.74
Config ABCRP 2222	44.26	0.62	24.75
LatentMaid 2222	34.09	0.08	3.04

Table 28: Training and caching costs for Scenario 1 (cached latents), EFLOPs. We cache ImageNet (1,281,167 images) twice, including horizontal flips.

VAE Configuration	VAE Train + Cache	With DiT-B/2	With DiT-XL/2
Config A	46.49	60.62	119.40
Config ABCRP	45.86	59.99	118.77
Config ABCRP 2222	44.88	59.01	117.79
LatentMaid 2222	34.17	48.30	107.08

Table 29: Scenario 2: Encode on the fly. All calculations in EFLOPs

VAE configuration	VAE train + online encoding	With DiT-B/2	With DiT-XL/2
Config A	73.81	87.93	146.71
Config ABCRP	71.93	86.06	144.84
Config ABCRP 2222	69.01	83.15	141.93
LatentMaid 2222	37.13	51.26	110.04