# Taming Sampling Perturbations with Variance Expansion Loss for Latent Diffusion Models

Supplementary Material

## A. Analyze: Why Gaussian Priors Are Unnecessary in Latent Diffusion.

To further understand why an explicit Gaussian prior is not required in latent diffusion models, we begin by revisiting the standard variational autoencoder (VAE) formulation. A variational autoencoder models data likelihood through a latent variable $z$:

$$p_\theta(x) = \int p_\theta(x \mid z)\, p(z)\, \mathrm{d}z, \tag{16}$$

where $p(z)$ is the latent prior, typically chosen as $\mathcal{N}(0, I)$ for tractability. Since the true posterior $p_\theta(z \mid x)$ is intractable, the encoder learns an approximation $q_\phi(z \mid x)$, leading to the evidence lower bound (ELBO):

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] - \mathrm{KL}(q_\phi(z \mid x) \,\|\, p(z)) \,. \tag{17}$$

The Gaussian prior $p(z) = \mathcal{N}(0, I)$ serves two purposes: (1) it regularizes the latent space, preventing the encoder from overfitting to individual samples, and (2) it allows analytical evaluation of the KL term, enabling stable optimization. Hence, the Gaussian assumption in VAEs is not merely a modeling choice but a mathematical necessity for tractable variational inference.

In latent diffusion models, the situation is fundamentally different. The latent variable $z_0$ (obtained from a tokenizer or encoder) is further diffused through a forward noising process:

$$q(z_t \mid z_{t-1}) = \mathcal{N}(\sqrt{\alpha_t} z_{t-1}, (1 - \alpha_t) I), \tag{18}$$

and the model learns the reverse transitions $p_\theta(z_{t-1} \mid z_t)$. This process defines an *implicit prior* over $z_0$:

$$p_\theta(z_0) = \int p(z_T) \prod_{t=1}^{T} p_\theta(z_{t-1} \mid z_t)\, \mathrm{d}z_{1:T}, \tag{19}$$

where $p(z_T) = \mathcal{N}(0, I)$ is only the noise prior at the terminal step. The marginal distribution $p_\theta(z_0)$ over the encoder's latent space is therefore *learned* by the diffusion model itself rather than fixed a priori. Consequently, constraining $q_\phi(z_0 \mid x)$ to follow a Gaussian distribution is both unnecessary and potentially harmful, as it restricts the expressiveness and robustness of the latent space learned through diffusion.

In summary, the key distinction between VAEs and latent diffusion models lies in where and how the latent prior is defined. While VAEs rely on an explicit, analytically specified Gaussian prior to regularize the latent distribution, latent diffusion models instead learn an implicit prior through the denoising trajectory and its associated score (or velocity) field. As a consequence, enforcing a predefined Gaussian constraint on the encoder's output is not only unnecessary, but can also distort the intrinsic geometry of the latent manifold, reduce its expressiveness, and ultimately hinder the diffusion model's ability to learn a faithful, data-driven prior in the latent space.

## B. More Training Details

For diffusion models trained in latent spaces aligned with DINOv2 [18] representations [29, 33], it is well known that, despite their strong performance, they tend to suffer from training instabilities under long optimization schedules. In practice, this often manifests as sudden loss spikes in the later stages of training, after which the optimization rarely recovers—a phenomenon that has been widely reported in the community. To mitigate this issue, RAE [33] employs a learning rate schedule that linearly decays from $2.0 \times 10^{-4}$ to $2.0 \times 10^{-5}$ with a constant warmup of 40 epochs. In our experiments, we observed that using the Muon optimizer [11] substantially alleviates this issue. Therefore, for all long-horizon training runs in this work, we adopt Muon as our default optimizer.

## C. Details of the 2D toy example

We largely follow the dataset construction protocol of Karras et al. [12], with one important modification: since our experiments do not make use of any class-dependent effects, we restrict the data distribution to a single class.

More specifically, we represent the fractal-like structure of the data by a Gaussian mixture $\mathcal{M}_{\mathbf{c}} = \big(\{\phi_i\}, \{\boldsymbol{\mu}_i\}, \{\boldsymbol{\Sigma}_i\}\big)$, where $\phi_i$, $\boldsymbol{\mu}_i$, and $\boldsymbol{\Sigma}_i \in \mathbb{R}^{2\times 2}$ denote the mixture weight, the mean, and the covariance matrix of component $i$, respectively. This parameterization admits closed-form expressions for both the density and its score, which enables exact computation and visualization without any further approximations. For a fixed class $\mathbf{c}$, the data density can be written as

$$p_{\text{data}}(\mathbf{x} \mid \mathbf{c}) = \sum_{i\in\mathcal{M}_{\mathbf{c}}} \phi_i \, \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), \tag{20}$$

where the two-dimensional Gaussian density is given by

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{2\pi\sqrt{\det(\boldsymbol{\Sigma})}} \exp\Big( -\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\Big). \tag{21}$$

Adding isotropic Gaussian noise of standard deviation $\sigma$ to $p_{\text{data}}(\mathbf{x} \mid \mathbf{c})$ corresponds to convolving it with a Gaussian kernel, which yields a family of smoothed densities $p(\mathbf{x} \mid \mathbf{c}; \sigma)$ parameterized by the noise level:

$$p(\mathbf{x} \mid \mathbf{c}; \sigma) = \sum_{i\in\mathcal{M}_{\mathbf{c}}} \phi_i \, \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}^*_{i,\sigma}), \quad \text{with} \quad \boldsymbol{\Sigma}^*_{i,\sigma} = \boldsymbol{\Sigma}_i + \sigma^2 \mathbf{I}. \tag{22}$$

The corresponding score function admits the closed-form expression

$$\nabla_{\mathbf{x}} \log p(\mathbf{x} \mid \mathbf{c}; \sigma) = \frac{\sum_{i\in\mathcal{M}_{\mathbf{c}}} \phi_i \, \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}^*_{i,\sigma}) \, (\boldsymbol{\Sigma}^*_{i,\sigma})^{-1}(\boldsymbol{\mu}_i - \mathbf{x})}{\sum_{i\in\mathcal{M}_{\mathbf{c}}} \phi_i \, \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}^*_{i,\sigma})}. \tag{23}$$

To obtain a thin, tree-shaped structure, we design $\mathcal{M}_{\mathbf{c}}$ by starting from a single main "branch" and recursively splitting it into smaller sub-branches. Each branch segment is represented by 8 anisotropic Gaussian components. The subdivision is repeated 6 times; after each split, we downscale the corresponding mixture weights $\phi_i$ and introduce small random perturbations to the lengths and orientations of the two child branches. This procedure produces $127 \times 8 = 1016$ components for the class considered in our experiments. Following the normalization guidelines of Karras et al. [12], we choose the coordinate system so that the mean of $p_{\text{data}}$ (marginalized over $\mathbf{c}$) is zero and the standard deviation along each axis is $\sigma_{\text{data}} = 0.5$.

**Models and training details.** We implement both the tokenizer and the denoiser (vector-field) networks as 8-layer ReLU MLPs (hidden dim 512). To make the latent space directly visualizable, we fix its dimensionality to two. Concretely, the tokenizer encoder maps a two-dimensional input point $\boldsymbol{x} \in \mathbb{R}^2$ to a two-dimensional latent code $\boldsymbol{z} \in \mathbb{R}^2$, and the decoder maps it back to the data space:

$$\boldsymbol{z} = f_{\text{enc}}(\boldsymbol{x}; \boldsymbol{\theta}_{\text{enc}}) \in \mathbb{R}^2, \tag{24}$$

$$\hat{\boldsymbol{x}} = f_{\text{dec}}(\boldsymbol{z}; \boldsymbol{\theta}_{\text{dec}}) \in \mathbb{R}^2. \tag{25}$$

The tokenizer is trained using a standard reconstruction objective with KL regularization or VE loss:

$$\mathcal{L}_{\text{rec}} = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\big[\, \|\boldsymbol{x} - f_{\text{dec}}(f_{\text{enc}}(\boldsymbol{x}))\| \big]. \tag{26}$$

In Figure 1, the baseline uses a KL coefficient of $10^{-3}$, while VE loss uses a coefficient of $10^{-2}$.

For the diffusion model, we adopt the flow-matching formulation, which provides a particularly simple and effective way to learn continuous-time generative dynamics. We define a bridging trajectory between a simple base distribution $p_0$ and the data distribution $p_{\text{data}}$ as

$$\boldsymbol{Z}_t = (1-t)\, \boldsymbol{Z}_0 + t\, \boldsymbol{X}, \quad t \in [0,1], \tag{27}$$

where $\boldsymbol{Z}_0 \sim p_0$ (we use a standard Gaussian $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$) and $\boldsymbol{X} \sim p_{\text{data}}$. The corresponding ground-truth velocity is simply

$$\dot{\boldsymbol{Z}}_t = \frac{\mathrm{d}\boldsymbol{Z}_t}{\mathrm{d}t} = \boldsymbol{X} - \boldsymbol{Z}_0. \tag{28}$$

Flow-matching models directly learn the time-dependent vector field $v_{\boldsymbol{\theta}}$ that approximates this velocity along the trajectory, using the loss

$$\mathcal{L}_{\text{flow}} = \mathbb{E}_{t\sim\mathcal{U}(0,1),\, \boldsymbol{Z}_t}\big[\|v_{\boldsymbol{\theta}}(\boldsymbol{Z}_t, t) - \dot{\boldsymbol{Z}}_t\|_2^2\big], \tag{29}$$

where $v_{\boldsymbol{\theta}}$ denotes the predicted instantaneous velocity and $\dot{\boldsymbol{Z}}_t$ is the ground-truth time derivative of the path defined above.

Both the tokenizer and the flow-matching model are trained with Adam for 200k iterations using batch size 4096, learning rate $10^{-3}$ with a schedule following AutoGuidance. At sampling time, we uses an Euler solver with 20 steps:

$$\frac{\mathrm{d}\boldsymbol{Z}_t}{\mathrm{d}t} = v_{\boldsymbol{\theta}}(\boldsymbol{Z}_t, t), \tag{30}$$

starting from $\boldsymbol{Z}_0 \sim p_0$ and evolving from $t = 0$ to $t = 1$ using a standard explicit Euler sampler with $N = 20$ uniform time steps. Denoting $t_k = k/N$ and $\Delta t = 1/N$, the sampler update reads

$$\boldsymbol{Z}_{t_{k+1}} = \boldsymbol{Z}_{t_k} + \Delta t\, v_{\boldsymbol{\theta}}(\boldsymbol{Z}_{t_k}, t_k), \quad k = 0, \dots, N-1, \tag{31}$$

and the final sample in data space is obtained by decoding the terminal latent state,

$$\hat{\boldsymbol{x}} = f_{\text{dec}}(\boldsymbol{Z}_{t_1}). \tag{32}$$

This setup keeps the model and training procedure minimal while allowing us to directly inspect both the learned latent representation and the generative trajectories in two dimensions.

## D. Visual Comparison

We provide additional visual examples of our method on *ImageNet* $256 \times 256$. Consistent with Figure 3 and Table 4, all samples are generated with a classifier-free guidance (CFG) scale of $1.45$ and a CFG interval of $[0.13, 1]$. Representative visual results are shown in Figures 4 and 5.

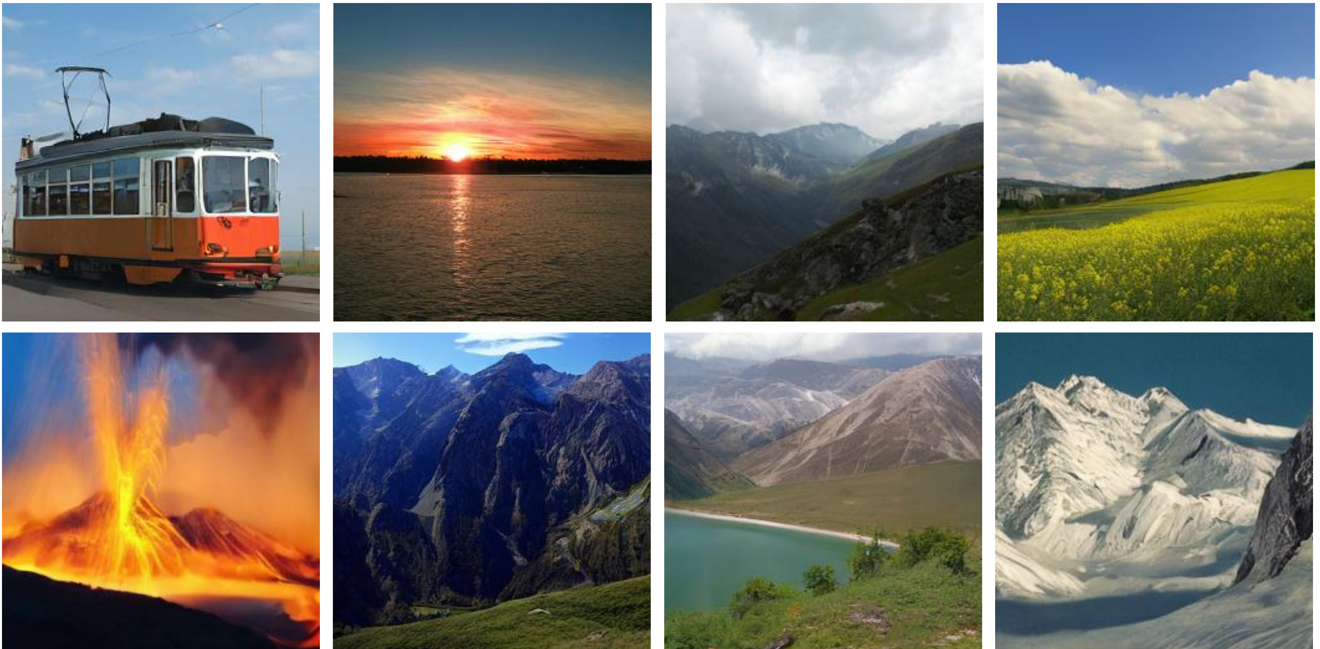

Figure 4. Visualization Results. Examples of class-conditional generation on *ImageNet* 256×256

Figure 5. Visualization Results. Examples of class-conditional generation on *ImageNet* 256×256