Latent Diffusion Inversion Requires Understanding the Latent Space

Supplementary Material

A. Approximation of Local Distortion

An pseudo-code to efficiently compute local distortion using randomized singular value decomposition is provided in Algo. 1. Noticeably, this code computing Jacobian–vector products using jvp/vjp packages in pytoch without explicitly computing the Jacobian matrix. In our experiments, finite-difference is also a good approximation of jvp when automatic differentiation is unavailable

Algorithm 1 Matrix–Free Randomized SVD for Top-k Singular Values of $J_D(z)$

*[t]

Require: Decoder mean map $D : \mathcal{Z} \to \mathcal{X}$; latent point $z \in \mathcal{Z}$; target rank k; oversampling p; power passes q; flag USE_JVP$\in$ {TRUE, FALSE}; finite-difference step h

Ensure: Approx. top-k singular values $\{s_i\}_{i=1}^{k}$ of $J_D(z)$ and log–volume $\sum_{i=1}^{k} \log s_i$

$d \leftarrow \dim(\mathcal{Z}), \quad m \leftarrow \dim(\mathcal{X}), \quad \ell \leftarrow \min(k+p,\, d)$

define decoder–Jacobian oracles at z:
$\mathcal{J}(V) = J_D(z)V \in \mathbb{R}^{m\times r}$ **via** JVP if USE_JVP=TRUE, **else** central differences: $[(D(z+hv_j) - D(z-hv_j))/(2h)]_j$
$\mathcal{J}^\top(Y) = J_D(z)^\top Y \in \mathbb{R}^{d\times r}$ **via** VJP (reverse-mode autodiff)

Draw $V_0 \sim \mathcal{N}(0, I_{d\times\ell})$; $\quad V \leftarrow \mathrm{qr}(V_0)$ $\quad \triangleright\, V \in \mathbb{R}^{d\times\ell}$

for $t = 1$ to q **do** $\quad \triangleright$ optional power iteration on $G = J_D^\top J_D$
  $Y \leftarrow \mathcal{J}(V) \in \mathbb{R}^{m\times\ell}$; $\quad G \leftarrow \mathcal{J}^\top(Y) \in \mathbb{R}^{d\times\ell}$
  $V \leftarrow \mathrm{qr}(G)$
end for

$Y \leftarrow \mathcal{J}(V) \in \mathbb{R}^{m\times\ell}$; $\quad Q \leftarrow \mathrm{qr}(Y) \in \mathbb{R}^{m\times\ell}$

$T \leftarrow \mathcal{J}^\top(Q) \in \mathbb{R}^{d\times\ell}$

$\{\hat{s}_i\}_{i=1}^{\ell} \leftarrow \mathrm{svdvals}(T)$; **sort** $\hat{s}_i$ in descending order

$s_i \leftarrow \hat{s}_i$ for $i = 1, \ldots, k$ $\quad \triangleright$ top-k singular values of $J_D(z)$

return $\{s_i\}_{i=1}^{k}$, $\quad$ log–volume $= \sum_{i=1}^{k} \log s_i$

B. Variation of Filtering Ratio k

We evaluated masking ratios $k \in [10\%, 90\%]$ on ImageNet-1K(subset) of SimA [27] attack statistics. The results are summarized on Table 4

C. Datasests and splits

The dataset and member/non-member splits information are provided in Table 5.

D. Per–Dimension Influence via Hutchinson Estimator

A pseudocode of our method to compute the per-dimension influence on memorization via the Hutchinson Estimator is provided in Algo. 2.

Algorithm 2 Per–Dimension Influence via Hutchinson Estimator

Require: Decoder D, latent samples $\{z^{(n)}\}_{n=1}^{N}$ with $z^{(n)} \in \mathbb{R}^d$, number of Monte Carlo probes n_{mc}, stability constant $\varepsilon > 0$

Ensure: Matrix PerDim $\in \mathbb{R}^{N\times d}$ containing per–dimension log–influence for each $z^{(n)}$

for $n = 1$ to N **do** $\quad \triangleright$ Process each latent code independently
  $z \leftarrow z^{(n)}$; enable gradient on z
  $x \leftarrow D(z)$; $\quad x \in \mathbb{R}^m$ $\quad \triangleright$ Decoder output (flattened)
  diag_est $\leftarrow \mathbf{0} \in \mathbb{R}^d$ $\quad \triangleright$ Accumulator for $\mathrm{diag}(J_D^\top J_D)$
  for $j = 1$ to n_{mc} **do** $\quad \triangleright$ Hutchinson probes
    $v \sim \mathcal{N}(\mathbf{0}, I_m)$ $\quad \triangleright$ Random probe in output space
    $g \leftarrow J_D(z)^\top v$ $\quad \triangleright$ Compute VJP via reverse–mode autodiff
    diag_est $\leftarrow$ diag_est $+ g \odot g$ $\quad \triangleright$ Elementwise square and accumulate
  end for
  diag_est $\leftarrow$ diag_est$/n_{\mathrm{mc}}$ $\quad \triangleright$ Monte Carlo average
  per_dim_log $\leftarrow \frac{1}{2}\log(\text{diag\_est} + \varepsilon)$ $\quad \triangleright$ Log–compressed influence
  PerDim$[n, :] \leftarrow$ per_dim_log
end for
return PerDim

E. Experiments on LDMs with VQ-GAN as the encoder

For experiments on MNIST, CIFAR-10 and CelebA, we used a vanilla β-VAE. For experiments on ImageNet, Pokémon, COCO-2017-Val, and Flickr30k, we employed Stability AI's autoencoders trained with **Adversarial (GAN) and Perceptual (LPIPS) losses**. We also test our masking strategy on VQ-GAN [10](discrete, large geometry change). The original LDM implementation used the bottleneck layer $\hat{z}$ before quantization; since we need the decoder side as well, we approximate $\frac{\partial D(\hat{z})}{\partial \hat{z}} \approx \frac{\partial D(z_q)}{\partial z_q}$ after quantization. We trained a VQ-GAN from scratch for CIFAR-10 and MNIST and used a pre-trained one[2] for ImageNet-1K. Training details, such as the number of training steps, are kept consistent with the other experiments. Table 6 shows that our method consistently looks for the most-memorizing dimension, which corresponds to the per-dimensional distortion.

[2] https://ommer-lab.com/files/latent-diffusion/vq-f8.zip

Table 4. AUC performance of SimA attack under different masking ratios k on ImageNet-1K (subset). For comparison, the last column presents a reverse-masking setting where the top 90% most-memorizing dimensions are removed. Retaining only the bottom 10% renders the attacks almost ineffective.

Masking Ratio k	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	**0.9 (Rev)**
AUC (↑)	69.70	70.77	71.68	72.32	72.64	72.54	71.95	70.65	68.91	65.99	**54.67**
TPR@1%FPR (↑)	3.96	4.68	5.64	7.06	7.79	8.09	8.62	9.04	8.46	6.78	**2.21**

Table 5. Datasets and splits used for our experiments. **VAE** denotes the **VAE** of LDMs. **LM** denotes **Diffusion Part of LDMs**

Model	**Member**	**Held-out**	**Pre-trained (VAE)**	**Pre-trained(LM)**	**Splits**	**Resolution**	**Cond.**
Latent Diffusion Models	MNIST	MNIST	No	No	30k/30k	28	–
	CIFAR-10	CIFAR-10	No	No	25k/25k	32	–
	CelebA	CelebA	No	No	30k/30k	64	–
	ImageNet-1k(train)	ImageNet-1K(Val)	Yes	No	100k/100k	256	class
Stable Diffusion V1-4	Pokémon	Pokémon	Yes	Yes	416/417	512	text
	COCO2017-Val	COCO2017-Val	Yes	Yes	2.5k/2.5k	512	text
	Flickr30k	Flickr30k	Yes	Yes	10k/10k	512	text

Table 6. **LDMs with VQGAN as the encoder**. Performance (AUC %) comparison on different datasets. Numbers in bold indicate improvement.

Method	MNIST	CIFAR-10	ImageNet-1K
SimA (After Filter)	83.69 (**+2.7**)	86.27 (**+3.2**)	67.38 (**+3.85**)

F. Latent diffusion attack by smoothing high-frequency components

In this section, we apply the method proposed by Lian et al. [19] to latent diffusion models (LDMs), specifically Stable Diffusion. Although the method was originally developed for data-domain diffusion models, we extend it to the latent domain using the following formulation.

Loss: The attack statistics can be expressed as, for $\varepsilon \sim \mathcal{N}(0, I)$:

$$\text{Loss} = \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} z_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon,\, t)\|, \tag{9}$$

where $\epsilon_\theta(\cdot)$ indicates the predicted noise. According to Appendix B of Lian et al. [19]. We derive

$$z_0^{\text{target}} = \frac{z_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon}{\sqrt{\bar{\alpha}_t}}, \tag{10}$$

$$z_0 = \frac{z_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta(\sqrt{\bar{\alpha}_t} z_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon,\, t)}{\sqrt{\bar{\alpha}_t}}. \tag{11}$$

To smooth the high-frequency components, we should map latent codes z to the image x. Then, $x_0 = D(z_0)$ and $x_0^{\text{target}} = D(z_0^{\text{target}})$.

PIA: Similarly, it can be expressed as:

$$\text{PIA} = \left\|\epsilon_\theta(z_0, 0) - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}\, z_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta(z_0, 0),\, t\right)\right\|, \tag{12}$$

where $\epsilon_\theta(z_0, 0)$ represents the noise prediction for z_0. According to Appendix B of Lian et al. [19], it can be converted to

$$z_0^{\text{target}} = \frac{z_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta(z_0, 0)}{\sqrt{\bar{\alpha}_t}}, \tag{13}$$

$$z_0 = \frac{z_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta(\sqrt{\bar{\alpha}_t} z_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta(z_0, 0),\, t)}{\sqrt{\bar{\alpha}_t}}. \tag{14}$$

Likewise, $x_0 = D(z_0)$ and $x_0^{\text{target}} = D(z_0^{\text{target}})$.

A high-frequncy filter is defined the same as Li et al. [18],

$$\mathcal{F}(x) = \text{IFFT}(\text{FFT}(x) \odot \beta(r)), \tag{15}$$

where FFT and IFFT denotes the *Fast Fourier Transform* and *Inverse Fast Fourier Transform*. $\odot$ denotes element-wise multiplication, and $\beta_{i,t}(r)$ is a mask that keeps the low-frequency components and smooths the high-frequency components. Specifically,

$$\beta(r) = \begin{cases} s, & \text{if } r > r_t, \\ 1, & \text{otherwise.} \end{cases} \tag{16}$$

where s is frequency-dependent scaling factor, r denotes the radius of frequency-domain, and r_t is the high-frequency threshold radius. Then the attack statistics for both Loss and PIA methods become:

$$\|\mathcal{F}(x_0) - \mathcal{F}(x_0^{\text{target}})\|. \tag{17}$$

Under the assumption that latent-space diffusion models inherit the same learning dynamics as data-domain diffusion models, one would expect the proposed high-frequency

filtering technique to be equally effective in both settings. Contrary to this expectation, our experiments show that the method decreases the attack performance for Stable Diffusion (Table 7). This divergence suggests that the mapping from the data domain to the latent domain—typically implemented through a nonlinear VAE encoder—introduces non-trivial distortions that alter the frequency structure relevant for membership inference. Consequently, careful consideration of this domain transformation is essential when adapting techniques developed for data-domain models.

G. More image examples on different local distortion

More examples of images in the high (low) distortion region are provided in Fig. 4. In a standard VAE architecture, the latent space is a bottleneck, meaning $k < d$. If $x = D(z)$ is purely deterministic, the generated data does not span the full $\mathbb{R}^d$ space. Instead, it lies strictly on a k-dimensional manifold embedded within the d-dimensional ambient space.

Because of this, $p(x)$ is a degenerate distribution in $\mathbb{R}^d$ (it has zero volume) and the standard change of variables formula fails. However, if we evaluate the density strictly on the manifold with respect to the k-dimensional Hausdorff measure, we must use the area formula. The metric tensor induced on the latent space by the mapping D is $J_D(z)^T J_D(z)$.The density on the manifold is [4, 12]:

$$p(x) = p(z) \left(\det \left(J_D(z)^T J_D(z)\right)\right)^{-1/2}$$

Taking the logarithm gives:

$$\log p(x) = \log p(z) - \frac{1}{2} \log \underbrace{\det \left(J_D(z)^T J_D(z)\right)}_{\text{Distortion(z)}} \quad (18)$$

According to [24, 26, 30], the log-likelihood estimation in a generative model is highly biased toward the input complexity, where the model assigns more density to complex samples. Based on Eq. 18, this insight explains the positive correlation between visual complexity and the distortion of a sample in Fig. 2 and Fig. 4. Hence, the distribution shifts significantly from the data space to the latent space.

H. Architecture details of all LDM instances

CIFAR-10. We train a convolutional VAE at 32×32 RGB resolution with 3 input channels, base width 128, and a 4-channel latent space. The encoder consists of strided convolutions and residual blocks with GroupNorm and SiLU activations, downsampling $32 \to 16 \to 8$; the decoder mirrors this structure. The VAE is optimized for 120 epochs on the member subset using AdamW (learning rate 2×10^{-4}, $\beta = (0.9, 0.999)$, weight decay 10^{-4}), with an ℓ_1 reconstruction term and a KL divergence term weighted by $\beta_{\text{KL}} = 10^{-2}$. On the frozen latent space, we train a DDPM-style UNet with 128 base channels, channel multipliers $(1, 2, 2, 2)$, two residual blocks per resolution level, dropout 0.1, and GroupNorm. The diffusion model uses a linear noise schedule with $T = 1000$ steps and $\beta \in [10^{-4}, 2 \times 10^{-2}]$, and is trained with AdamW (same hyperparameters) for 2048 epochs with batch size 128.

CelebA. For CelebA, we first center-crop faces to 140×140, resize to 64×64, and apply random horizontal flips and per-channel normalization to $[-1, 1]$. We train a VAE with the same architectural configuration (3 input channels, base width 128, 4-channel latent space) for 120 epochs using AdamW (learning rate 2×10^{-4}, $\beta = (0.9, 0.999)$, weight decay 10^{-4}), but with a smaller KL weight $\beta_{\text{KL}} = 10^{-3}$. The latent diffusion model is a UNet with 224 base channels and channel multipliers $(1, 2, 3, 4)$, two residual blocks per level, dropout 0.1, and a linear noise schedule with $T = 1000$ diffusion steps and $\beta \in [10^{-4}, 2 \times 10^{-2}]$. It is trained with AdamW (same optimizer settings) for 512 epochs with batch size 256.

ImageNet-1K. We train a class-conditional latent diffusion model on ImageNet-1K using the pretrained Stable Diffusion VAE (`sd-vae-ft-mse`) as a frozen encoder–decoder (with default $\beta_{KL} = 1e-6$). Input images are resized and center-cropped to 256×256 RGB, augmented with random horizontal flips, and normalized to $[-1, 1]$. The VAE deterministically encodes images into $4 \times 32 \times 32$ latents ($8\times$ spatial downsampling), which are scaled by 0.18215 before diffusion and rescaled back when decoding.

On this latent space, we train a UNet with 4 input/output channels, 2 residual blocks per resolution, and block widths $(192, 384, 576, 960)$, using cross-attention layers with a learned class embedding of dimension 512. Class conditioning is implemented via an embedding table over the 1,000 ImageNet labels, passed as the UNet cross-attention context. The diffusion process uses $T = 1000$ steps with a linear noise schedule $\beta \in [1.5 \times 10^{-3}, 1.95 \times 10^{-2}]$.

We construct a balanced training subset with 50 images per class (50k images total) from the ImageNet-1K training split and use the standard validation split for evaluation and qualitative sampling. The UNet is optimized for 600 epochs with AdamW (learning rate 10^{-4}, $\beta = (0.9, 0.999)$, zero weight decay) and a cosine learning-rate schedule with 1,000 warm-up steps. Training is performed with mixed-precision (FP16) and distributed data parallelism with an effective per-device batch size of 21.

Fine-tuning on Stable diffusion. We reuse the Stable Diffusion V1-4 architecture for all stable diffusion experiments.

Table 7. Attack performance of four baseline methods used to evaluate memorization of **Stable Diffusion**.

Method	Pokémon (%)			MS-COCO (%)			Flickr (%)		
	AUC↑	ASR↑	TPR@1%FPR↑	AUC↑	ASR↑	TPR@1%FPR↑	AUC↑	ASR↑	TPR@1%FPR↑
Loss	92.03	85.47	40.77	83.26	75.80	13.48	63.13	59.74	2.35
Loss (+high freq. filter)	84.39	78.40	19.90	73.46	68.08	4.32	57.52	55.81	1.61
PIA	94.93	90.52	32.85	92.51	85.72	24.76	68.88	64.61	2.67
PIA (+high freq. filter)	89.81	84.27	19.90	73.32	67.72	4.96	57.37	55.61	1.43

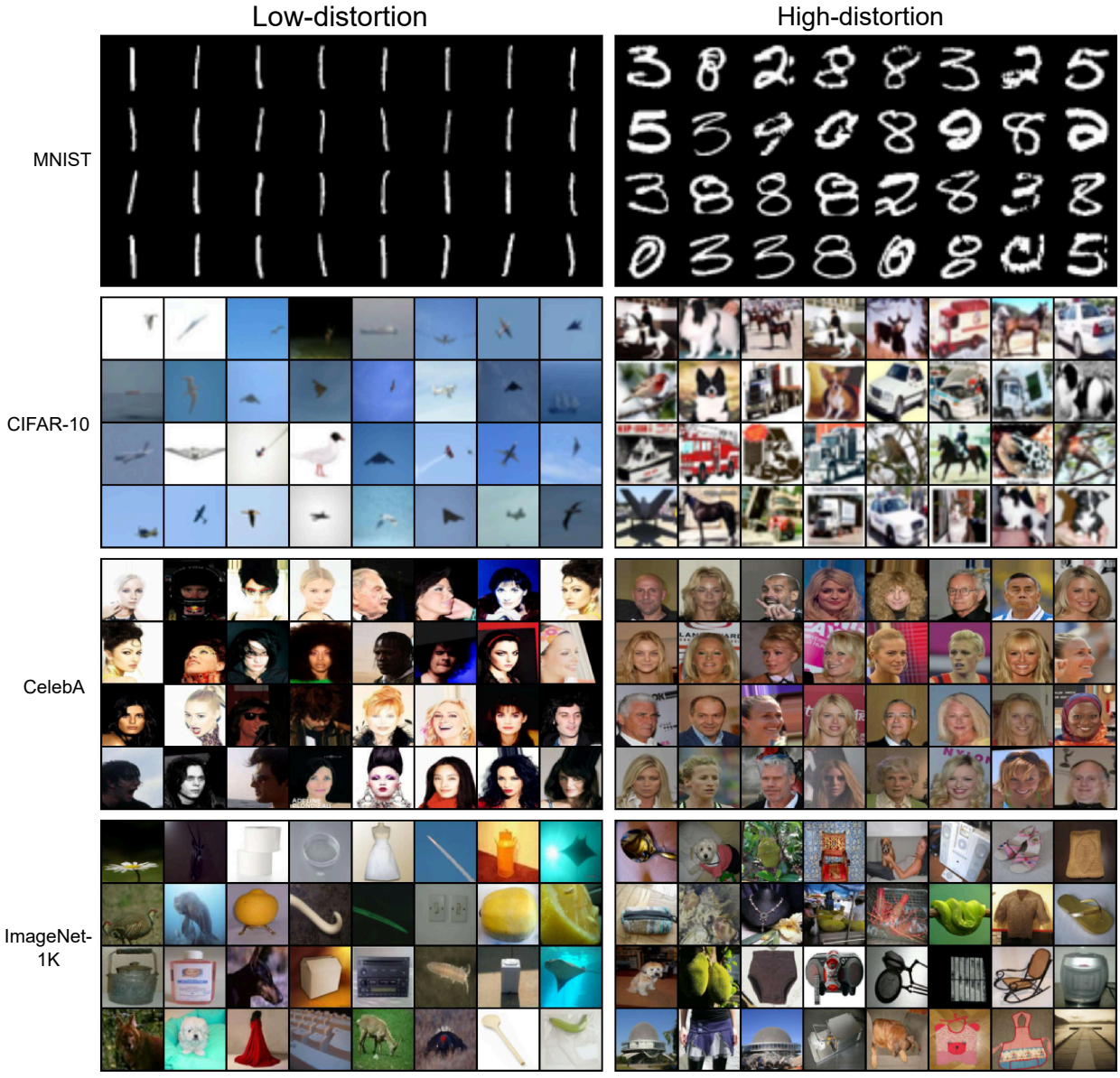

Figure 4. More examples of images in high (low) distortion region