# GENERATIVE LATENT ZONING NETWORKS

**Anonymous authors**
Paper under double-blind review




Figure 1: Class-conditional image generation and editing on `ImageNet256×256`; image editing requires *no task-specific training*. Editing references are outlined in red. See § F for more examples.

## ABSTRACT

Recent advances in AI have been driven largely by generative models such as diffusion models (DMs), autoregressive (AR) models, and generative adversarial networks (GANs). However, each paradigm has inherent limitations in generation efficiency, training stability, latent-space flexibility, or other desirable properties. We propose a new class of generative models called `Generative Latent Zoning Networks` (`G-LZNs`). `G-LZNs` map each sample to a *latent zone* with learnable boundaries and decode latents within the zone back to the sample. By design, `G-LZNs` simultaneously provide six desirable properties: **(1) One-step generation,** unlike DMs and AR; **(2) Stable training** with reconstruction-based losses, avoiding competing objectives such as those in GANs; **(3) Flexible latent** dimensionality and latent-to-data mapping, unlike AR, which lacks an explicit latent space, and DMs, whose latent space is constrained by their formulation; **(4) Native inversion** from samples to latent space, unlike GANs and AR; **(5) Likelihood bounds,** unlike GANs; and **(6) Flexible architecture,** unlike normalizing flows, which require invertible architectures. Their compact latent space and native inversion further enable **image editing without task-specific training**, including class changes, class and latent interpolation, fine-grained attribute editing, super-resolution, and inpainting. Crucially, `G-LZNs` achieve these properties without sacrificing generation quality. Among one-step generative models trained from scratch, **`G-LZNs` set new state-of-the-art image generation results**, achieving FID 1.88 on `CIFAR10`, and $\text{FDr}^6$ 2.49 on `ImageNet256×256`. Code and models will be released upon acceptance.



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.

Table 1: Comparison of representative generative models. A checkmark indicates that the property is supported *natively by the original formulation*, without auxiliary models or post-hoc procedures. See § 2 for detailed definitions or categorizations of each property.

| Model | One-step generation | Stable training | Flexible latent | Native inversion | Likelihood bounds | Flexible architecture |
|---|---|---|---|---|---|---|
| Diffusion Models (DMs) / Flow Matching (FM) | | ✓ | | ✓ | ✓ | ✓ |
| Autoregressive Models (AR) | | ✓ | | | ✓ | ✓ |
| Generative Adversarial Networks (GANs) | ✓ | | ✓ | | | ✓ |
| Variational Autoencoders (VAEs) | ✓ | | ✓ | ✓ | ✓ | ✓ |
| Normalizing Flows (NFs) | ✓ | ✓ | | ✓ | ✓ | |
| Energy-Based Models (EBMs) | | | | | | ✓ |
| Consistency Models (CMs) | ✓ | | | | | ✓ |
| Inductive Moment Matching (IMM) | ✓ | | | | | ✓ |
| MeanFlow (MF) | ✓ | | | | | ✓ |
| Drifting Models | ✓ | | ✓ | | | ✓ |
| **Generative Latent Zoning Networks (G-LZNs)** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

## 1 Introduction

Recent advances in AI have been driven largely by *generative models*, which learn an unknown data distribution from observed samples and generate new samples from it. Prominent examples include diffusion models (DMs) (Ho et al., 2020; Song & Ermon, 2019; Sohl-Dickstein et al., 2015), autoregressive models (ARs) (Tian et al., 2024; Sun et al., 2024), and generative adversarial networks (GANs) (Goodfellow et al., 2020). However, existing generative models exhibit different trade-offs. For example, diffusion models and ARs typically require many sequential steps for generation and lack flexible latent spaces, while GANs can be difficult to train and lack explicit likelihoods. Overall, no existing paradigm simultaneously satisfies all the desirable properties summarized in Tab. 1.

A large body of work has sought to mitigate these limitations in a post-hoc manner, for example by improving training algorithms (Arjovsky et al., 2017), applying post-hoc acceleration or refinement techniques (Liu et al., 2025b;a), or combining multiple generative paradigms (Xiao et al., 2021; Vahdat et al., 2021); we discuss these approaches in more detail in § A. While effective, such methods often introduce additional complexity into the training or inference pipelines and do not eliminate limitations that arise from the underlying model design itself. Instead, we ask:

*Can we design a new generative model to achieve state-of-the-art quality without these trade-offs?*

Tab. 1 summarizes the desiderata we seek to satisfy. This goal is ambitious: many of these properties have traditionally been viewed as competing objectives (Xiao et al., 2021).

In this paper, we introduce a new class of generative models called Generative Latent Zoning Networks (G-LZNs) (Fig. 2). A G-LZN consists of an *encoder* and a *decoder*. Unlike conventional encoders that map each sample to a fixed embedding (Makhzani et al., 2015) or a simple posterior distribution (Kingma & Welling, 2013), the G-LZN encoder maps each sample to a *zone* in latent space through a *latent computation process*. Each zone has a learnable boundary, and the posterior distribution within the zone is matched to the latent distribution restricted to that zone. The decoder then maps latent points from the zone back to the corresponding sample. G-LZNs are trained from scratch without relying on any pretrained generative models.

By design, G-LZNs satisfy all the desiderata in Tab. 1: **(1) One-step generation:** generation requires only a single forward pass of a random latent through the decoder; **(2) Stable training:** training uses *only reconstruction losses*, without competing objectives or surrogate losses; **(3) Flexible latent:** the latent dimensionality is decoupled from the data dimensionality, and the latent-to-data mapping is unconstrained; **(4) Native inversion:** the latent computation process maps samples back to latents; **(5) Likelihood lower bounds:** the likelihood of a given sample can be estimated natively; and **(6) Flexible architecture:** no architectural constraints are imposed on the encoder or decoder, unlike in some existing generative models (Dinh et al., 2014; Rezende & Mohamed, 2015).

In addition, thanks to the compact latent space and native inversion, **G-LZNs support six image-editing tasks *without task-specific training***: class changes, latent interpolation, class interpolation, fine-grained attribute editing, image super-resolution, and image inpainting (Fig. 1 and § F.2).

Importantly, these desirable properties do not sacrifice generation quality. **G-LZNs set new image-generation records** among one-step methods trained without pretrained generative models, achieving FID 1.88 on CIFAR10 and FDr$^6$ 2.49 on ImageNet256×256, all in one step. They even

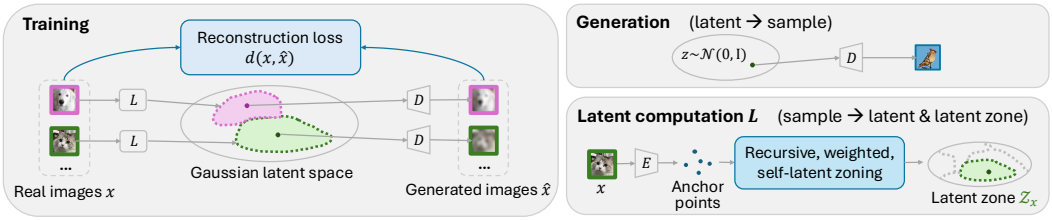


Figure 2: Overview of G-LZN. **Left: Training.** Each real image $x$ is mapped by the *latent computation* process $L$ to a latent within its corresponding *latent zone* in a Gaussian latent space, and the *decoder* $D$ maps the latent back to a generated image $\hat{x}$. Training uses only reconstruction losses between $x$ and $\hat{x}$. **Top right: Generation.** A latent $z \sim \mathcal{N}(0, \mathbf{I})$ from the Gaussian prior is mapped by the decoder to a generated sample *in a single step*. **Bottom right: Latent computation $L$.** The *encoder* $E$ maps $x$ to a set of anchor points, from which *recursive, weighted, self-latent zoning* constructs a *latent zone* $\mathcal{Z}_x$ with flexible, learnable boundaries and samples a latent within the zone.

outperform many diffusion and AR models requiring up to 2,000 steps and/or over 10× larger models (Tab. 2), as well as methods relying on pretrained generative models (Tabs. 9 and 10).

## 2 Background and Desiderata

In this section, we discuss key desiderata for generative models, why they matter, and where existing paradigms fall short. For details of individual methods, we refer readers to the corresponding papers.

**One-step generation.** Inference latency and cost are critical for deployment. However, dominant generative paradigms require multi-step generation: DMs rely on iterative denoising, while ARs generate token by token. One might argue that multiple steps allow harder samples to receive more computation. However, in DMs and ARs, these steps are dictated by artificial intermediate states such as noise levels or tokens, rather than by sample difficulty. Alternative mechanisms that allocate compute where needed, such as adaptive layers (Graves, 2016) or latent-space iterations (Hao et al., 2024), are better suited for this purpose and are compatible with one-step generative models.

**Stable training.** Training stability means smooth within-run optimization and consistency across random seeds. It is important at scale (Sauer et al., 2023). Empirical stability also depends on architecture and hyperparameters; we focus only on two objective-level criteria. First, training should avoid *competing objectives*: GANs optimize opposing generator and discriminator objectives, which can cause instability (Arjovsky & Bottou, 2017), while VAEs balance reconstruction and KL regularization associated with posterior collapse (Bowman et al., 2016). Second, it should avoid *self-dependent surrogate targets*: CMs and MF use stop-gradient targets (Song et al., 2023; Geng et al., 2025), which have been associated with instability (Geng et al., 2025; He et al., 2026).

**Flexible latent.** A latent space enables controllable generation through operations on representations mapped to the data space (Lin et al., 2020; Radford et al., 2015). We seek two properties. **(a) Flexible dimensionality:** latent dimensionality should be decoupled from data dimensionality, enabling compact, semantically structured representations (Radford et al., 2015). **(b) Flexible mapping:** given the latent and data distributions, the formulation should not prescribe their transport, allowing mappings suited to different applications (Lin et al., 2020).

Many paradigms violate these properties. Data-space DMs, FMs, NFs, CMs, IMM, and MF require equal latent and data dimensionality, violating (a). DMs, FMs, CMs, IMM, and MF also prescribe an interpolation between the two distributions, fixing the target transport and limiting (b). ARs and EBMs define no latent space. Using a pretrained autoencoder latent space (Rombach et al., 2022) does not remove these constraints: the model's latent remains constrained relative to the autoencoder-induced "data space," and is learned separately rather than through generative training.

**Native inversion.** Among generative models with latent spaces, some can directly map samples to latents, enabling applications including image editing (Shen et al., 2020), image labeling (Lin et al., 2020), and representation learning (Radford et al., 2015). GANs lack a native inverse mapping, making inversion difficult or requiring extra modifications (Xia et al., 2022).

**Likelihood bounds.** Likelihood measures a sample's probability under the generated distribution. Although not necessarily correlated with perceptual quality (Theis et al., 2015), likelihood can be useful for anomaly detection (Ren et al., 2019). GANs lack likelihood estimation.

**Flexible architecture.** Ideally, a generative formulation should impose minimal constraints on network architecture, allowing new architectural advances to be adopted directly. However, NFs (Dinh et al., 2014; Rezende & Mohamed, 2015) require invertible transformations with tractable Jacobians.

# 3 Generative Latent Zoning Networks (G-LZNs)

## 3.1 Overall Framework

In this subsection, we rethink from scratch what a generative modeling framework should look like to satisfy the desiderata in § 2.

**Encoder-decoder framework.** To support *flexible latent* and *native inversion*, we start from the encoder-decoder framework used in existing generative models (Kingma & Welling, 2013; Sohl-Dickstein et al., 2015). Given a sample $x$, a latent computation process $L$ maps it to $L(x)$; given a latent $z$, a decoding process $D$ maps it to a sample $D(z)$. We deliberately call $L$ a *latent computation process*, rather than an encoder, as it need not be a single neural network.

**Decoder.** To satisfy *one-step generation* and *flexible architecture*, $D$ can be an unconstrained feed-forward network mapping a latent $z$ to a sample $x$ in one pass. We call $D$ the *decoder*.

**Loss function.** To satisfy *stable training*, the simplest choice is to use only the reconstruction loss[1]

$$\mathbb{E}_{x\sim\mathcal{D}}\left(d\left(x, D\left(L\left(x\right)\right)\right)\right), \tag{1}$$

where $\mathcal{D}$ denotes the training dataset and $d$ is a distance function (e.g., $\ell_1$). This objective is derived directly from data, without surrogate targets or competing terms.

**Challenge: latent computation $L$.** The key challenge is designing $L$. If implemented as an unconstrained neural network, the framework reduces to a standard autoencoder, whose induced latent distribution $L(x)$ for $x \sim \mathcal{D}$ can be arbitrary and difficult to sample for generation. Existing approaches address this issue with techniques that sacrifice one or more desiderata; see § A. Designing $L$ to satisfy all desiderata is therefore non-trivial. We present our solution next.

## 3.2 Detailed Designs on Latent Computation

### 3.2.1 Latent Zoning Networks (LZNs)

We briefly review latent zoning networks (LZNs) (Lin et al., 2026b), on which our method builds. LZNs unify generation, representation learning, and classification through a shared latent space, with modality-specific encoders and decoders composed for different tasks. We refer readers to the paper (Lin et al., 2026b) for details and focus only on the latent zoning mechanism relevant here.

**Latent zoning.** Given $n$ samples $\mathcal{D} = \{x_1, \ldots, x_n\}$, LZN defines a stochastic latent computation $C$ that produces $z_1, \ldots, z_n = C(\mathcal{D})$, where $\mathcal{Z}_i \triangleq \text{Supp}(z_i)$ is the *latent zone* deterministically associated with $x_i$. The computation satisfies two key properties: **(P1) Gaussian latent distribution:** the marginal latent distribution satisfies $z \sim \text{Uniform}\{z_1, \ldots, z_n\} = \mathcal{N}(0, \mathbf{I})$; and **(P2) Latent zone separation:** different samples occupy disjoint zones, $\mathcal{Z}_i \cap \mathcal{Z}_j = \emptyset$ for $i \neq j$.[2] Together, P1 and P2 imply that each zone has Gaussian probability mass $1/n$, and the latent distribution within each zone is exactly the Gaussian prior restricted to that zone: $p(z \mid x_i) = n\mathcal{N}(z; 0, \mathbf{I})\,\mathbb{1}[z \in \mathcal{Z}_i]$.

LZN realizes this latent computation by first mapping each sample to an *anchor point* $a_i = E(x_i)$ with a deterministic encoder $E$, and then using flow matching (FM) (Liu et al., 2022; Lipman et al., 2022) to transform the anchor distribution into latent zones. Specifically, let $\pi_0 = \mathcal{N}(0, \mathbf{I})$ and $\pi_1(s) = \frac{1}{n}\sum_{i=1}^{n}\delta_{a_i}(s)$,[3] with intermediate distributions $\pi_t$ induced by the linear interpolation $\varphi(s_0, s_1, t) = (1-t)s_0 + ts_1$ for $s_0 \sim \pi_0$ and $s_1 \sim \pi_1$. The corresponding FM velocity field is $V(s,t) = \frac{\sum_{i=1}^{n}(a_i - s)\exp\left(-\frac{(s-ta_i)^2}{2(1-t)^2}\right)}{(1-t)\sum_{i=1}^{n}\exp\left(-\frac{(s-ta_i)^2}{2(1-t)^2}\right)}$. For each anchor $a_i$, we initialize $s_{1-g} = (1-g)a_i + g\epsilon$,

[1] We use "reconstruction loss" broadly to also include methods that match group-level statistics. For simplicity, we focus on sample-level losses in this formulation and extend it to group-level losses in § 3.4.

[2] Strictly speaking, P2 is approximate under the finite smoothing ($g$) used in LZN; see Lin et al. (2026b) for the analysis. For simplicity, we adopt the idealized setting in which P2 holds exactly in all theoretical analysis.

[3] $\delta$ denotes the Dirac delta function.

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ and $g$ is a small constant,[4] and define inverse flow matching (IFM) by backward integration: $\text{IFM}\left(a_i, \epsilon; a_1, \ldots, a_k\right) \triangleq s_{1-g} + \int_{\tau=1-g}^{0} V(s_\tau, \tau)\mathrm{d}\tau$. The latent computation is then $z_i = C\left(\mathcal{D}\right)_i \triangleq \text{IFM}\left(a_i, \epsilon; a_1, \ldots, a_k\right)$.

Importantly, the latent zone boundaries are implicitly determined by learnable anchor points, and the latent computation is differentiable, allowing end-to-end optimization through the encoder.

**Challenges.** At first glance, we could use LZN's latent computation directly by setting $L\left(x_i\right) \triangleq C\left(\mathcal{D}\right)_i$. If the loss in Eq. (1) reaches zero, then by construction the generated distribution $D\left(z\right)$ for $z \sim \mathcal{N}\left(0, \mathbf{I}\right)$ exactly matches the empirical distribution Uniform $\{\mathcal{D}\}$, yielding a valid generative model for the training data. However, this is computationally impractical: computing $C\left(\mathcal{D}\right)$ requires $\mathcal{O}\left(n^2\right)$ time and memory, prohibitive for modern pretraining datasets. LZN therefore applies latent zoning only within minibatches (Lin et al., 2026b). Consequently, each sample's latent zone depends on its minibatch and varies across iterations; more importantly, zones assigned to samples in different minibatches can overlap, breaking the argument above. Thus, the original LZN uses latent zoning only to improve existing generative models rather than as a standalone model.

**Overview of our solutions.** We first introduce *self-latent zoning*, which removes the quadratic dependence on the number of samples, reducing the complexity from $\mathcal{O}\left(n^2\right)$ to $\mathcal{O}\left(n\right)$ and enabling minibatch training (§ 3.2.2). However, the large number of anchor points still makes the method computationally and memory intensive. We therefore introduce two additional techniques, *recursive latent zoning* and *weighted latent zoning*, to make the method practical (§ 3.2.3).

### 3.2.2 Self-Latent Zoning

Instead of computing latent zones jointly across all samples, we compute each sample's zone independently. For each sample $x$, the encoder outputs $k$ anchors: $a_1, \ldots, a_k = E\left(x\right)$, and we define the latent of $x$ as the latent associated with the first anchor: $L\left(x\right) \triangleq C\left(\{a_1, \ldots, a_k\}\right)_1$. This reduces the full-dataset latent-zoning complexity from $\mathcal{O}\left(n^2\right)$ to $\mathcal{O}\left(n\right)$ and enables standard minibatch training for Eq. (1) due to independent per-sample computation.

The trade-off is that latent zones of different samples are no longer guaranteed to be disjoint, violating Property P2 in § 3.2.1. However, we will show in § 3.3 that minimizing the reconstruction loss implicitly encourages zone separation, allowing the model to remain a valid generative model.

### 3.2.3 Recursive and Weighted Latent Zoning

While self-latent zoning removes the need to jointly process all $n$ samples, it remains impractical at scale. The latent computation for each sample costs $\mathcal{O}\left(k\right)$, while ensuring that each sample occupies Gaussian probability mass $1/n$ requires $k = n$. Thus, computing the latent zones for all samples still costs $\mathcal{O}\left(n^2\right)$. We next introduce two techniques to substantially reduce this cost.

**Recursive latent zoning.** The operation in § 3.2.2, $C\left(\{a_1, \ldots, a_k\}\right)_1 = \text{IFM}\left(a_1, \epsilon; a_1, \ldots, a_k\right)$, maps $\epsilon \sim \mathcal{N}\left(0, \mathbf{I}\right)$ to a zone with $1/k$ of the Gaussian mass. We recursively apply it to $\epsilon$, partitioning the mass by $k$ at each recursion. Specifically, the encoder now outputs $lk$ anchors, $\{a_{ij}\}_{i\in[l],\, j\in[k]} = E\left(x\right)$, with $a_{i1}, \ldots, a_{ik}$ used at recursion $i$. Starting from $\epsilon_1 \sim \mathcal{N}\left(0, \mathbf{I}\right)$, we compute

$$\epsilon_{i+1} = \text{IFM}\left(a_{i1}, \epsilon_i; a_{i1}, \ldots, a_{ik}\right), \quad i = 1, \ldots, l, \qquad\qquad L\left(x\right) \triangleq \epsilon_{l+1}.$$

The resulting latent zone occupies $1/k^l$ of the Gaussian probability mass, with per-sample time and memory complexity $\mathcal{O}\left(l \cdot k\right)$. Setting $k^l = n$ gives $l = \log_k n$ and hence complexity $\mathcal{O}\left(k \log_k n\right)$. For constant $k$, this becomes $\mathcal{O}\left(n \log n\right)$ for the full dataset, improved substantially from $\mathcal{O}\left(n^2\right)$ without recursive latent zoning. We call $l$ *the number of latent zoning layers*.

**Weighted latent zoning.** The above construction assigns equal weights to anchors $a_{i1}, \ldots, a_{ik}$ in the FM endpoint distribution (§ 3.2.1): $\pi_1(s) = \frac{1}{k}\sum_{j=1}^{k} \delta_{a_{ij}}(s)$. We instead use non-uniform weights to reduce the probability mass assigned to the first anchor. Specifically, we introduce

[4] FM requires the endpoint distributions to have full support. Since $\pi_1$ is Dirac delta mixture that lacks full support, LZN instead starts from the smoothed distribution $\pi_{1-g} = (1-g)\pi_1 + g\mathcal{N}(0, \mathbf{I})$.

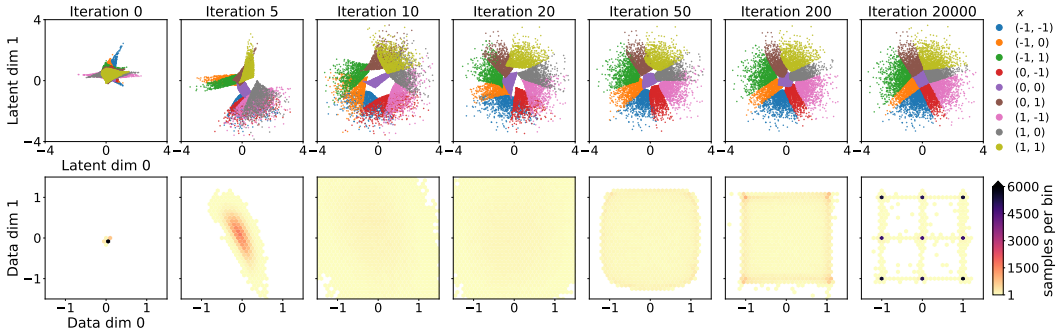


Figure 3: A toy example of training G-LZN on nine data points $\{-1, 0, 1\}^2$ with a two-dimensional latent space. **Top:** latent zones for each sample $x$, visualized with 1,000 draws of $\epsilon_1$ in $L(x)$. **Bottom:** heatmap of 50,000 generated samples $D(z)$. Early in training, the zones overlap and the generated distribution differs from the data distribution. As training progresses, the zones separate, jointly cover the latent space, and form an interpretable structure aligned with the data points, while the generated distribution approaches the data distribution. See § D for experimental details.

weights $w_1, \ldots, w_k$ with $w_j > 0$ and $\sum_{j=1}^{k} w_j = 1$, and define $\pi_1(s) = \sum_{j=1}^{k} w_j \delta_{a_{ij}}(s)$. The Gaussian mass of the first anchor's zone then changes from $1/k$ to ${w_1}^l$. With recursive latent zoning, the final zone occupies ${w_1}^l$ of the Gaussian mass, which we define as the *latent zone size* $S$. Setting $S = 1/n$ gives $l = \log_{1/w_1} n$ and per-sample complexity $\mathcal{O}\left(k \log_{1/w_1} n\right)$. Choosing $w_1 \ll 1/k$ therefore requires fewer recursion layers than uniform weighting, further improving efficiency.

**Summary.** The pseudocode of the final latent computation algorithm is provided in Alg. 1. Its key hyperparameters are $k$, $l$, and $\{w_i\}_{i \in [k]}$, whose effects are studied in § 4.4 and F. Fig. 3 visualizes the learned latent zones, which can take highly flexible shapes and be either bounded or unbounded. § A discusses the connections and differences between G-LZNs and existing generative models.

### 3.3 Theoretical Analysis

We first show that G-LZN exactly recovers the empirical training distribution at zero loss in Eq. (1). We model the decoder as a conditional output distribution $P_D(\cdot \mid z)$. Given $z$, let $\widehat{X} \sim P_D(\cdot \mid z)$ denote its output. A deterministic decoder is the special case $P_D(\cdot \mid z) = \delta_{D(z)}$. The expectation in Eq. (1) includes randomness from both the latent computation and decoder. For any sample $x$, let $P_{Z|x}$ denote the distribution induced by $L(x)$ and $\mathcal{Z}_x$ its latent zone. Let $P_Z = \mathcal{N}(0, \mathbf{I})$ be the latent prior, with density $p_Z$. By construction, $P_{Z|x} = P_Z(\cdot \mid \mathcal{Z}_x)$ and $P_Z(\mathcal{Z}_x) = {w_1}^l$.

**Theorem 1** (Exact recovery of the empirical distribution)**.** *Suppose that (i) $d(x, y) = 0$ if and only if $x = y$; (ii) the training samples $x_1, \ldots, x_n$ are pairwise distinct; (iii) the reconstruction loss in Eq. (1) is zero; and (iv) ${w_1}^l = 1/n$. Then, for $Z \sim P_Z$ and $\widehat{X} \sim P_D(\cdot \mid Z)$, we have $\widehat{X} \sim \frac{1}{n} \sum_{i=1}^{n} \delta_{x_i}$, i.e., the generated distribution exactly matches the empirical training distribution.*

See § B.1 for proof. Intuitively, zero reconstruction loss forces the conditional decoder distribution to concentrate on $x_i$ almost everywhere in its latent zone. Since the training samples are distinct, their latent zones must be disjoint almost everywhere. Each zone has Gaussian mass ${w_1}^l = 1/n$, so the $n$ zones jointly cover the latent distribution, yielding the empirical distribution after decoding.

Fig. 3 illustrates this process. Early training, overlapping zones cause distribution mismatch; reconstruction training improves the decoder and separates the zones, driving generation toward the data distribution. In practice, we seek generalization beyond the training set, as discussed in § 3.4.

Next, we show a lower bound on data likelihood. When $P_D(\cdot \mid z)$ admits a normalized, evaluable conditional density (for continuous data) or probability mass function (for discrete data) $p_D(x \mid z)$, the resulting marginal data density or probability mass function is $p_D(x) = \int p_D(x \mid z) p_Z(z) \mathrm{d}z$.

**Theorem 2** (Data log-likelihood lower bound)**.** *For any sample $x$, the log-likelihood satisfies*

$$\log p_D(x) \geq \mathbb{E}_{Z \sim P_{Z|x}} \left(\log p_D(x \mid Z)\right) + l \log w_1. \tag{2}$$

See § B.2 for proof. As in a VAE, the first term in Eq. (2) is an expected decoder log-likelihood that can be estimated by sampling from the latent computation. Unlike in a VAE, the latent-space KL

contribution is the known constant $\mathrm{KL}\left(P_{Z|x} \,\|\, P_Z\right) = -l \log w_1$, which is fixed by the latent-zone hyperparameters rather than learned through regularization.

Thm. 2 also gives a direct interpretation of our training loss in Eq. (1) as optimizing data likelihood.

**Corollary 1** (Reconstruction loss as optimizing data likelihood)**.** *Let $P_{\text{data}}$ be any data distribution, and let $\mathcal{L}_{\text{rec}}$ denote the objective in Eq. (1) for $x \sim P_{\text{data}}$, including model randomness. Suppose there exist a constant $\beta > 0$ and a function $c(x)$ independent of the model parameters such that*

$$\log p_D(x \mid z) = c(x) - \beta\, \mathbb{E}_{\widehat{X} \sim P_D(\cdot|z)} \left( d\left(x, \widehat{X}\right)\right). \tag{3}$$

*Then, whenever the expectations are well-defined,*

$$\mathbb{E}_{X \sim P_{\text{data}}} \left(\log p_D(X)\right) \geq -\beta \mathcal{L}_{\text{rec}} + \mathbb{E}_{X \sim P_{\text{data}}} \left(c(X)\right) + l \log w_1. \tag{4}$$

*Consequently, when $w_1$ and $l$ are fixed, minimizing the reconstruction loss maximizes this lower bound on the average data log-likelihood.*

See § B.3 for proof. For example, if $P_D(\cdot \mid z)$ is a fixed-variance Gaussian with mean $D(z)$, squared Euclidean reconstruction distance satisfies Eq. (3) with $\beta = 1/(2\sigma^2)$ and parameter-independent $c(x)$. Unlike Thm. 1, Corollary 1 holds *without requiring a finite dataset or* ${w_1}^l = 1/n$.

### 3.4 Implementation

**Controlling memorization and generalization via latent zone size $S$.** Thm. 1 shows that $S = 1/n$ recovers the empirical distribution. To encourage generalization and novelty, we set $S \ll 1/n$; the techniques in § 3.2.3 efficiently support very small $S$. This choice is also supported by Corollary 1. § F.6 shows that G-LZNs can smoothly interpolate between memorization and generalization by varying $S$, unlike existing models, where this trade-off lacks an explicit control knob and is governed implicitly by factors such as model and dataset size (Gu et al., 2023; Ye et al., 2026).

**Reconstruction loss.** Unlike DMs/RFs, whose theoretically grounded objectives are more constrained (Lee et al., 2024), G-LZNs can use any distance $d$ in Eq. (1) with $d(x, y) = 0$ iff $x = y$ (Thm. 1). $\ell_1$ and $\ell_2$ already work (§ F.9.1). Motivated by successes of using pretrained embeddings to improve generation (Deng et al., 2026; Yu et al., 2025), we develop two embedding-based losses:

*(1) Sample-wise improved perceptual loss.* Standard perceptual loss (i.e., LPIPS) (Zhang et al., 2018) computes $\ell_1$ or $\ell_2$ reconstruction losses on intermediate features from a pretrained embedding model. Inspired by Drifting Models (Deng et al., 2026), we further add reconstruction losses on feature statistics and downsampled feature maps. See § C.1.1 for details.

*(2) Group-wise Fréchet loss.* Recent work (Yang et al., 2026a) improves generation by minimizing Fréchet distance between real and generated embeddings, approximating each distribution as Gaussian. This approximation can be inaccurate for multimodal data such as `ImageNet`. We instead compute Fréchet losses within groups sharing conditions such as class labels, yielding a *group-wise reconstruction loss*. To reduce the cost of evaluating many groups per iteration, we develop an approximation comparable in cost to the original Fréchet loss. See § C.1.2 for details.

**Compatible with pretrained autoencoders.** For efficiency, high-resolution models often use a pretrained autoencoder (AE) $\{\mathcal{E}, \mathcal{D}\}$ (Rombach et al., 2022). G-LZNs can optimize either *in AE latent space* as $\mathbb{E}_{x \sim \mathcal{D}} \left(d\left(\mathcal{E}(x), D\left(L\left(\mathcal{E}(x)\right)\right)\right)\right)$, or *in pixel space* as $\mathbb{E}_{x \sim \mathcal{D}} \left(d\left(x, \mathcal{D}\left(D\left(L\left(\mathcal{E}(x)\right)\right)\right)\right)\right)$. Latent diffusion models usually can only optimize in AE latent space (Rombach et al., 2022), where better distribution matching may not improve decoded image quality. In contrast, G-LZNs can optimize end-to-end in pixel space while retaining latent-space efficiency.

**Architecture.** We use a ResNet-based U-Net common in diffusion models (Liu et al., 2022), remove skip connections, and use its encoder and decoder halves as our encoder and decoder. Although transformers often perform better empirically, we restrict experiments to ResNet to stress-test the generative formulation and leave other architectures to future work. See § C.2 for details.

**Conditional generation.** G-LZNs support conditional generation by providing the condition (e.g., a class label) to both the encoder and decoder. Accordingly, the loss in Eq. (1) becomes $\mathbb{E}_{x,c \sim \mathcal{D}} \left(d\left(x, D\left(L\left(x, c\right), c\right)\right)\right)$, where $c$ denotes the condition. See § C.2 for details.

Table 2: Among methods that do *not* use or initialize from pretrained generative checkpoints, G-LZNs achieve state-of-the-art one-step generation, and even surpass many multi-step methods and far larger models; Tabs. 9 and 10 give more baselines. #Params counts the generator plus any pretrained decoder or tokenizer. **Bold** marks the best among one-step (NFE 1) models, underline the best at any NFE. Sources for every number are given in Tabs. 11 and 12.

| | Method | NFE ↓ | Param | FID ↓ | IS ↑ |
|---|---|---|---|---|---|
| GANs | StyleGAN2 (2020b) | 1 | 20M | 8.32 | 9.21 |
| | StyleGAN2-ADA (2020a) | 1 | 20M | 2.92 | 9.83 |
| VAEs | NVAE (2021) | 1 | 52M | 23.50 | 7.18 |
| | CoVAE (2025) | 1 | 62M | 17.21 | — |
| NFs | Glow (2018) | 1 | 44M | 48.90 | 3.92 |
| | Residual Flow (2020) | 1 | 25M | 46.40 | — |
| DMs/FM | DDPM (2020) | 1000 | 36M | 3.17 | 9.46 |
| | Rectified Flow (2022) | 127 | 62M | 2.58 | 9.60 |
| | Score SDE (2020b) | 2000 | 63M | 2.38 | 9.83 |
| | PFGM (2022) | 110 | 107M | 2.35 | 9.68 |
| | Score SDE (deep) (2020b) | 2000 | 108M | 2.20 | 9.89 |
| | LSGM (2021) | 138 | 376M | 2.10 | — |
| | EDM (2022) | 35 | 56M | 1.97 | 9.84 |
| | PFGM++ (2023) | 35 | 56M | 1.91 | — |
| CMs | iCT (2023) | 1 | 56M | 2.83 | 9.54 |
| | | 2 | 56M | 2.46 | 9.80 |
| | iCT-deep (2023) | 1 | 98M | 2.51 | 9.76 |
| | | 2 | 98M | 2.24 | 9.89 |
| | sCT (2025) | 1 | 63M | 2.85 | — |
| | | 2 | 63M | 2.06 | — |
| Others | IGEBM (2020) | 60 | 19M | 40.60 | 6.02 |
| | MeanFlow (2025) | 1 | 55M | 2.92 | — |
| | Sphere Encoder (2026) | 1 | 311M | 35.67 | 6.70 |
| | | 6 | 311M | 2.34 | 10.20 |
| | IMM (2025) | 1 | 56M | 3.20 | — |
| | | 2 | 56M | 1.98 | — |
| | **G-LZNs (ours)** | 1 | 57M | **1.88** | **10.33** |

(a) Unconditional CIFAR10

| | Method | Space | NFE | #Params | $FD^6_r$ ↓ | FID↓ | IS↑ | Prec↑ | Rec↑ |
|---|---|---|---|---|---|---|---|---|---|
| DMs / FM | JiT-L (2026) | pix | 50×2×2 | 459M | 10.73 | 2.59 | 288.5 | 0.79 | 0.59 |
| | SiT-XL/2 (2024) | lat | 250×2 | 675M+49M | 8.44 | 2.12 | 256.7 | 0.81 | 0.60 |
| | PixNerd-XL (2025a) | pix | 100×2 | 1.0B | 5.01 | 2.10 | 318.8 | 0.81 | 0.59 |
| | JiT-H (2026) | pix | 50×2×2 | 953M | 7.66 | 1.97 | 296.0 | 0.78 | 0.63 |
| | REG (2025) | lat | 250×2 | 685M+49M | 4.64 | 1.54 | 302.9 | 0.78 | 0.62 |
| | SiT-XL/2 + REPA (2025) | lat | 250×2 | 675M+49M | 5.45 | 1.42 | 306.1 | 0.80 | 0.65 |
| | LightningDiT-XL/2 (2025) | lat | 250×2 | 675M+70M | 4.57 | 1.42 | 294.3 | 0.80 | 0.64 |
| | DDT-XL (2025b) | lat | 250×2 | 675M+49M | 5.70 | 1.26 | 309.3 | 0.79 | 0.66 |
| | REPA-E (2025) | lat | 250×2 | 676M+49M | 3.04 | 1.17 | 298.3 | 0.79 | 0.66 |
| | RAE + $DiT_{DH}$-XL/2 (2025a) | lat | 50×2 | 839M+415M | 3.26 | 1.16 | 261.0 | 0.77 | 0.67 |
| AR | VAR-d30 (2024) | disc | 10×2 | 2B+65M | 6.70 | 1.97 | 304.6 | 0.82 | 0.59 |
| | MAR-L (2024) | lat | 256×2×100 | 478M+38M | 6.68 | 1.80 | 293.4 | 0.80 | 0.60 |
| | FlowAR-H (2024) | lat | 50×2 | 1.9B+38M | 6.13 | 1.68 | 274.1 | 0.80 | 0.62 |
| | MAR-H (2024) | lat | 256×2×100 | 942M+38M | 5.61 | 1.56 | 299.5 | 0.80 | 0.62 |
| | MAR-L + DeTok (2026b) | lat | 256×2×100 | 478M+86M | 5.49 | 1.39 | 306.2 | 0.81 | 0.62 |
| | BAR-L (2026a) | disc | 256×2×4 | 1.1B+303M | 3.57 | 1.01 | 281.9 | 0.77 | 0.68 |
| MF | pMF-L (2026) | pix | 1 | 410M | 9.09 | 2.72 | 261.7 | **0.81** | 0.56 |
| | pMF-H (2026) | pix | 1 | 935M | 6.87 | 2.29 | 267.2 | 0.80 | 0.59 |
| | iMF-XL/2 (2026) | lat | 1 | 610M+49M | 8.39 | 1.82 | 278.9 | 0.78 | 0.63 |
| | iMF-XL/2 (2026) | lat | 2 | 610M+49M | 7.48 | 1.61 | 289.1 | 0.79 | 0.63 |
| Others | Drifting, L/2 (2026) | lat | 1 | 463M+49M | 10.92 | 1.53 | 257.2 | 0.79 | 0.63 |
| | W-Flow, B/2 (2026) | lat | 1 | 133M+49M | 11.62 | 1.52 | 271.8 | 0.80 | 0.62 |
| | Drifting, L/16 (2026) | pix | 1 | 465M | 10.51 | 1.43 | 305.8 | **0.81** | 0.60 |
| | W-Flow, L/2 (2026) | lat | 1 | 463M+49M | 10.55 | 1.35 | 272.5 | 0.79 | 0.63 |
| | W-Flow, XL/2 (2026) | lat | 1 | 679M+49M | 9.51 | 1.29 | 265.4 | 0.76 | **0.67** |
| | **G-LZNs (ours)** | lat | 1 | 230M+49M | 3.62 | **1.20** | 338.7 | 0.79 | 0.52 |
| | **G-LZNs (ours)** | pix | 1 | 182M | **2.49** | 1.74 | **339.5** | **0.81** | 0.57 |

(b) Class-conditional ImageNet256×256

## 4 Experiments and Analysis

**Datasets.** We evaluate on conditional ImageNet256×256 (Deng et al., 2009) and unconditional CIFAR10 (Krizhevsky et al., 2009). ImageNet is a standard image-generation benchmark, and many recent works evaluate primarily on it (Tian et al., 2024; Deng et al., 2026). Its diversity and resolution make it a strong scalability test. We also evaluate on CIFAR10. Despite lower resolution, CIFAR10 has been studied and optimized for years, making further improvements challenging.

**Architectures.** On ImageNet256×256, we evaluate with and without pretrained autoencoders (Rombach et al., 2022), showing G-LZNs support both pixel- and latent-space generation.

**Metrics.** We report standard metrics: FID (Heusel et al., 2017), sFID (Nash et al., 2021), Inception Score (IS) (Salimans et al., 2016), precision, and recall (Kynkäänniemi et al., 2019). We also report $FDr^6$ (Yang et al., 2026a), which better aligns with perceptual image quality than FID.

Next, we show that G-LZNs achieve state-of-the-art generation (§ 4.1), support image editing (§ 4.2), and train stably (§ 4.3), followed by ablations (§ 4.4).

### 4.1 State-of-the-Art Image Generation

**G-LZNs achieve state-of-the-art image generation.** Tab. 2 shows that G-LZNs achieve state-of-the-art one-step image generation among methods that do not use or initialize from pretrained generative models. Remarkably, *with a single generation step*, G-LZNs outperform *every multi-step method listed, including substantially larger ones*, on FID and IS on CIFAR10, and $FDr^6$ and IS on ImageNet256×256. They also surpass many methods using pretrained generative models (see Tabs. 9 and 10). Generated images are shown in Figs. 1, 4 and 5.

**G-LZNs generalize well beyond the training set.** In § F.5, we show qualitatively that G-LZNs generate images distinct from the training set, and quantitatively that their samples are more novel than both the real validation set and strong baselines. These results confirm that the strong performance of G-LZNs is *not* due to training-image replication.

### 4.2 Image Editing without Task-Specific Training

G-LZNs support diverse image-editing applications without task-specific training (Fig. 1 and § F.2):

- **Changing image classes** by decoding image latents with desired class conditions. Figs. 1, 6 and 7 show transformations across very different classes (e.g., cars and animals) while preserving

realism. Even details like water reflections change accordingly. We can also create imaginative yet realistic scenes, such as a large strawberry floating on water (Fig. 7).

- **Class and latent interpolation** to smoothly vary object position, background, color, distance, and type (Figs. 1, 8 and 9). This works even across very different classes, such as a scenic landscape and a zebra (Fig. 8). Although pixel-space changes are smooth, semantic class can transition relatively sharply, keeping intermediate images realistic (e.g., dogs to cats in Fig. 8).
- **Editing image attributes via latent directions** enables fine-grained control over attributes such as object orientation, distance, environment, quantity, and age (Figs. 1 and 10). We obtain each direction by subtracting the mean latents of two groups with opposite attribute states, then add or subtract it from a query latent to continuously control the attribute.
- **Image super-resolution and inpainting** can be performed by encoding corrupted images into latent space and decoding the resulting latents back to images (§ F.2.5).

We emphasize that these experiments are qualitative demonstrations that G-LZNs learn a semantically structured and smooth latent space as a byproduct of generative training, rather than a claim of state-of-the-art editing; quantitative editing benchmarks are left to future work. We encourage readers to zoom in on the figures in § F.2 for more details.

### 4.3 Stable Training

**G-LZN training is stable within and across runs.** As validated in Fig. 20 with 5 runs, G-LZNs exhibit no loss/gradient spikes, remain stable post-convergence, and show low across-run variance throughout training. FIDs are also highly consistent: 1.83, 1.84, 1.88, 1.88, and 1.90.[5] In contrast, VAEs show loss spikes under the same setting (§ F.8), while GANs can degrade with prolonged training and exhibit substantial variance across random seeds (Lin et al., 2021).

### 4.4 Ablation Studies and Hyperparameter Analysis

We study key components and hyperparameters. Full results are in § F.6, F.8 and F.9.

**What makes G-LZNs work.** Replacing latent zoning with the VAE objective leads to worse performance and unstable training (§ F.8), while using only the Fréchet loss without sample-wise losses (§ C.1 and 3.4) leads to worse performance and image artifacts (§ F.9.1). These results highlight latent zoning (§ 3.2) as the key mechanism behind G-LZNs' success.

**Loss functions.** Simple $\ell_1$ or $\ell_2$ reconstruction losses without pretrained embeddings work (§ F.9.1). Our embedding-based reconstruction losses (§ 3.4) further improve performance (§ F.9.1 and F.9.2).

**Model size.** Larger models improve G-LZNs performance (§ F.9.3). However, even a small 45.8M model on ImageNet256×256 achieves FDr$^6$ 3.51, outperforming all but two baselines in Tab. 2b.

**Other hyperparameters.** G-LZNs are robust to latent dimension (§ F.9.7), zoning layers (§ F.9.5), number of anchors (§ F.9.6), and the use of a pretrained autoencoder (§ F.9.4). Latent zone size smoothly controls the generalization–memorization trade-off (§ F.6), unlike other generative models where it is implicitly determined by model and dataset size.

## 5 Discussions, Limitations, and Future Work

While much work addresses limitations of established generative paradigms, our results highlight a complementary direction: *designing new models with desirable properties by construction rather than through post-hoc fixes*. They suggest the generative-modeling design space remains far from exhausted and challenge the view that these desiderata must trade off. Future directions include:

- Our theory (Thm. 1) covers the idealized regime $S = 1/n$, while in practice we use $S \ll 1/n$. Corollary 1 narrows this gap, but full accounting of generalization properties remains open.
- Although G-LZNs work without pretrained embeddings, their best results currently benefit from them, as do many other generative models (Deng et al., 2026; Yu et al., 2025). An interesting direction is whether G-LZNs can match their current performance without external embeddings, potentially by leveraging their own encoder to learn suitable representations.
- The G-LZN framework supports different modalities, continuous or discrete tokens, and multi-step generation. Extending it to these settings is an interesting direction for future work.

[5] Throughout the paper, we report results from the run with the *median* FID.

## AI USE STATEMENT

We detail our use of AI in developing the ideas, implementation, and writing of this work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

**Ideas.** All ideas, including the core method, efficiency optimizations, and experimental design, originated from the authors, with one exception: the idea of sharing the covariance matrix across classes in the group-wise reconstruction loss (§ C.1.2) was suggested by AI.

**Implementation.** The initial working codebase of `G-LZNs` was written manually by the authors, including the core latent-zoning mechanism. AI was later used to help implement additional functionality, including more embedding models in the loss functions, efficiency optimizations, and additional evaluation metrics. All AI-generated code was reviewed by the authors and often required multiple rounds of manual revision to match the coding style of the original codebase and/or fix AI implementation bugs.

AI was also used to implement parts of the authors' research toolkit, including job and result monitoring and dashboards.

**Writing.** Most of the paper was first written by the authors and then iteratively polished by both AI and the authors for language, grammar, clarity, and conciseness. The following are the main exceptions.

Baseline results. We used AI to help collect and verify baseline metrics for the main result tables (Tabs. 2, 9 and 10). Specifically, we provided AI with a set of seed papers containing strong baselines and asked it to: (1) survey additional papers, including strong baselines published more recently; (2) cross-check numbers for the same method reported across different papers; (3) when possible, run the open-source code of baselines to verify reported parameter counts; (4) run important baselines with missing metrics, including evaluating the FDr[6], precision, and recall of W-Flow (Han et al., 2026), which had the strongest previously reported one-step FID on `ImageNet256×256`, using our evaluation pipeline;[6] and (5) document the source of every reported number in complete source tables (Tabs. 11 and 12).

The authors then manually verified each reported image quality metrics against its cited source. All numerical values were confirmed to be correct, although several source citations produced during this process were initially incorrect and were subsequently fixed by the authors.

We believe that, with appropriate human verification, AI assistance made this process more comprehensive and less error-prone than manually collecting all results from prior work. In particular, AI helped identify several strong baselines that had appeared very recently on arXiv and were initially missed by the authors, and it identified multiple inconsistencies in how prior papers reported the same results, which we document in Tabs. 11 and 12.

Ethics statement. We used AI to assist in drafting the ethics statement.

## ETHICS STATEMENT

This work develops a general-purpose generative modeling framework and evaluates it on standard public image benchmarks. Our experiments do not involve human subjects or the collection of new personal data. Like other image generative models, `G-LZNs` may inherit biases from their training data and may be misused to generate or manipulate misleading visual content. Its native inversion and image-editing capabilities may further facilitate such manipulation. We encourage appropriate safeguards, provenance mechanisms, and responsible deployment when applying these models to real-world or human-centered data.

## REPRODUCIBILITY STATEMENT

**Theory.** The assumptions, theorem statements, and proofs are provided in § B and 3.3.

[6] We confirmed that the FID measured by our pipeline closely matched the value reported in prior work, validating our evaluation setup.

**Experimental results.** A major source of irreproducibility in generative modeling is training instability (Goodfellow, 2016), where results can vary substantially within and across runs (§ 2). Improving training stability is exactly one of the main goals of this work: G-LZNs are designed to require only reconstruction losses, avoiding competing objectives. In § 4.3, we empirically verify that G-LZNs training is stable both within individual runs and across random seeds.

As in much prior work on generative models, the substantial training cost makes it impractical to repeat every experiment multiple times. To ensure that our main conclusions are robust, we run our main CIFAR10 experiment with five random seeds and confirm low variance across runs (§ 4.3). Throughout the paper, the reported metrics and generated images on CIFAR10 are taken from the run with the *median* FID among these five runs, rather than the best run, to avoid inflating performance due to randomness.

We provide experiemtal details in § D and E. Code and trained models will be released upon acceptance to facilitate reproducibility.

## REFERENCES


Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pp. 214–223. Pmlr, 2017.

Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Qinsheng Zhang, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, et al. ediff-i: Text-to-image diffusion models with an ensemble of expert denoisers. *arXiv preprint arXiv:2211.01324*, 2022.

Matthias Bauer and Andriy Mnih. Resampled priors for variational autoencoders. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 66–75. PMLR, 2019.

Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International conference on machine learning*, pp. 573–582. PMLR, 2019.

Piotr Bojanowski, Armand Joulin, David Lopez-Paz, and Arthur Szlam. Optimizing the latent space of generative networks. *arXiv preprint arXiv:1707.05776*, 2017.

Samuel Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of the 20th SIGNLL conference on computational natural language learning*, pp. 10–21, 2016.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis, 2019. URL https://arxiv.org/abs/1809.11096.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *2021 IEEE/CVF international conference on computer vision (ICCV)*, pp. 9630–9640. IEEE, 2021.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. Maskgit: Masked generative image transformer, 2022. URL https://arxiv.org/abs/2202.04200.

Hansheng Chen, Kai Zhang, Hao Tan, Leonidas Guibas, Gordon Wetzstein, and Sai Bi. pi-flow: Policy-based few-step generation via imitation distillation, 2026. URL https://arxiv.org/abs/2510.14974.

Ricky T. Q. Chen, Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. Residual flows for invertible generative modeling, 2020. URL https://arxiv.org/abs/1906.02735.

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 conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Mingyang Deng, He Li, Tianhong Li, Yilun Du, and Kaiming He. Generative modeling via drifting. *arXiv preprint arXiv:2602.04770*, 2026.

Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis, 2021. URL https://arxiv.org/abs/2105.05233.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Yilun Du and Igor Mordatch. Implicit generation and generalization in energy-based models, 2020. URL https://arxiv.org/abs/1903.08689.

Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12873–12883, 2021.

Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models, 2025. URL https://arxiv.org/abs/2410.12557.

Zhengyang Geng, Ashwini Pokle, William Luo, Justin Lin, and J. Zico Kolter. Consistency models made easy, 2024. URL https://arxiv.org/abs/2406.14548.

Zhengyang Geng, Mingyang Deng, Xingjian Bai, J. Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling, 2025. URL https://arxiv.org/abs/2505.13447.

Zhengyang Geng, Yiyang Lu, Zongze Wu, Eli Shechtman, J. Zico Kolter, and Kaiming He. Improved mean flows: On the challenges of fastforward generative models, 2026. URL https://arxiv.org/abs/2512.02012.

Partha Ghosh, Mehdi SM Sajjadi, Antonio Vergari, Michael Black, and Bernhard Schölkopf. From variational to deterministic autoencoders. *arXiv preprint arXiv:1903.12436*, 2019.

Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Xiangming Gu, Chao Du, Tianyu Pang, Chongxuan Li, Min Lin, and Ye Wang. On memorization in diffusion models. *arXiv preprint arXiv:2310.02664*, 2023.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

Jiaqi Han, Puheng Li, Qiushan Guo, Renyuan Xu, Stefano Ermon, and Emmanuel J. Candès. One-step generative modeling via wasserstein gradient flows, 2026. URL https://arxiv.org/abs/2605.11755.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

Valentin Hartmann, Anshuman Suri, Vincent Bindschaedler, David Evans, Shruti Tople, and Robert West. Sok: Memorization in general-purpose large language models. *arXiv preprint arXiv:2310.18362*, 2023.

Junxian He, Daniel Spokoyny, Graham Neubig, and Taylor Berg-Kirkpatrick. Lagging inference networks and posterior collapse in variational autoencoders. *arXiv preprint arXiv:1901.05534*, 2019.

Xiao He, Yang Li, Peizhen Zhang, Songtao Liu, Zhao Zhong, and Nannan Wang. Stabilizing, scaling & enhancing meanflow for large-scale diffusion distillation. *arXiv preprint arXiv:2605.17834*, 2026.

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. *Advances in neural information processing systems*, 30, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020. URL https://arxiv.org/abs/2006.11239.

Matthew D Hoffman, Matthew J Johnson, et al. Elbo surgery: yet another way to carve up the variational evidence lower bound. In *Workshop in advances in approximate Bayesian inference, NIPS*, volume 1, 2016.

Emiel Hoogeboom, Jonathan Heek, and Tim Salimans. Simple diffusion: End-to-end diffusion for high resolution images, 2023. URL https://arxiv.org/abs/2301.11093.

Emiel Hoogeboom, Thomas Mensink, Jonathan Heek, Kay Lamerigts, Ruiqi Gao, and Tim Salimans. Simpler diffusion (sid2): 1.5 fid on imagenet512 with pixel-space diffusion, 2025. URL https://arxiv.org/abs/2410.19324.

Minguk Kang, Jun-Yan Zhu, Richard Zhang, Jaesik Park, Eli Shechtman, Sylvain Paris, and Taesung Park. Scaling up gans for text-to-image synthesis, 2023. URL https://arxiv.org/abs/2303.05511.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 4396–4405. IEEE, 2019.

Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data, 2020a. URL https://arxiv.org/abs/2006.06676.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan, 2020b. URL https://arxiv.org/abs/1912.04958.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models, 2022. URL https://arxiv.org/abs/2206.00364.

Dongjun Kim, Yeongmin Kim, Se Jung Kwon, Wanmo Kang, and Il-Chul Moon. Refining generative process with discriminator guidance in score-based diffusion models, 2023. URL https://arxiv.org/abs/2211.17091.

Dongjun Kim, Chieh-Hsin Lai, Wei-Hsiang Liao, Naoki Murata, Yuhta Takida, Toshimitsu Uesaka, Yutong He, Yuki Mitsufuji, and Stefano Ermon. Consistency trajectory models: Learning probability flow ode trajectory of diffusion, 2024. URL https://arxiv.org/abs/2310.02279.

Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions, 2018. URL https://arxiv.org/abs/1807.03039.

Diederik P. Kingma and Ruiqi Gao. Understanding diffusion objectives as the elbo with simple data augmentation, 2023. URL https://arxiv.org/abs/2303.00848.

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

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. *Advances in neural information processing systems*, 32, 2019.

Sangyun Lee, Zinan Lin, and Giulia Fanti. Improving the training of rectified flows. *Advances in neural information processing systems*, 37:63082–63109, 2024.

Jiachen Lei, Keli Liu, Julius Berner, Haiming Yu, Hongkai Zheng, Jiahong Wu, and Xiangxiang Chu. There is no vae: End-to-end pixel-space generative modeling via self-supervised pre-training, 2026. URL https://arxiv.org/abs/2510.12586.

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.

Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise, 2026. URL https://arxiv.org/abs/2511.13720.

Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization, 2024. URL https://arxiv.org/abs/2406.11838.

Zengyi Li, Yubei Chen, and Friedrich T Sommer. Learning energy-based models in high-dimensional spaces with multiscale denoising-score matching. *Entropy*, 25(10):1367, 2023.

Shanchuan Lin, Ceyuan Yang, Zhijie Lin, Hao Chen, and Haoqi Fan. Adversarial flow models, 2026a. URL https://arxiv.org/abs/2511.22475.

Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. Pacgan: The power of two samples in generative adversarial networks. *Advances in neural information processing systems*, 31, 2018.

Zinan Lin, Kiran Thekumparampil, Giulia Fanti, and Sewoong Oh. Infogan-cr and modelcentrality: Self-supervised model training and selection for disentangling gans. In *international conference on machine learning*, pp. 6127–6139. PMLR, 2020.

Zinan Lin, Vyas Sekar, and Giulia Fanti. Why spectral normalization stabilizes gans: Analysis and improvements. *Advances in neural information processing systems*, 34:9625–9638, 2021.

Zinan Lin, Enshu Liu, Xuefei Ning, Junyi Zhu, Wenyu Wang, and Sergey Yekhanin. Latent zoning network: A unified principle for generative modeling, representation learning, and classification. *Advances in Neural Information Processing Systems*, 38:146239–146284, 2026b.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Enshu Liu, Xuefei Ning, Zinan Lin, Huazhong Yang, and Yu Wang. Oms-dpm: Optimizing the model schedule for diffusion probabilistic models. In *International Conference on Machine Learning*, pp. 21915–21936. PMLR, 2023.

Enshu Liu, Junyi Zhu, Zinan Lin, Xuefei Ning, Matthew B Blaschko, Shengen Yan, Guohao Dai, Huazhong Yang, and Yu Wang. Efficient expert pruning for sparse mixture-of-experts language models: Enhancing performance and reducing inference costs. *arXiv preprint arXiv:2407.00945*, 2024.

Enshu Liu, Qian Chen, Xuefei Ning, Shengen Yan, Guohao Dai, Zinan Lin, and Yu Wang. Distilled decoding 2: One-step sampling of image auto-regressive models with conditional score distillation, 2025a. URL https://arxiv.org/abs/2510.21003.

Enshu Liu, Xuefei Ning, Yu Wang, and Zinan Lin. Distilled decoding 1: One-step sampling of image auto-regressive models with flow matching, 2025b. URL https://arxiv.org/abs/2412.17153.

Enshu Liu, Junyi Zhu, Zinan Lin, Xuefei Ning, Shuaiqi Wang, Matthew Blaschko, Sergey Yekhanin, Shengen Yan, Guohao Dai, Huazhong Yang, et al. Linear combination of saved checkpoints makes consistency and diffusion models better. In *International Conference on Learning Representations*, volume 2025, pp. 10246–10282, 2025c.

Enshu Liu, Xuefei Ning, Yu Wang, and Zinan Lin. Ni sampling: Accelerating discrete diffusion sampling by token order optimization. *arXiv preprint arXiv:2604.18471*, 2026a.

Enshu Liu, Theodore Zhengde Zhao, Xuefei Ning, Zinan Lin, and Yu Wang. Ni sampling++: Optimizing token order for fast discrete diffusion sampling with reinforcement learning. In *Third Conference on Language Modeling*, 2026b.

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.

Cheng Lu and Yang Song. Simplifying, stabilizing and scaling continuous-time consistency models, 2025. URL https://arxiv.org/abs/2410.11081.

Yiyang Lu, Susie Lu, Qiao Sun, Hanhong Zhao, Zhicheng Jiang, Xianbang Wang, Tianhong Li, Zhengyang Geng, and Kaiming He. One-step latent-free image generation with pixel mean flows, 2026. URL https://arxiv.org/abs/2601.22158.

Eric Luhman and Troy Luhman. Knowledge distillation in iterative generative models for improved sampling speed. *arXiv preprint arXiv:2101.02388*, 2021.

Nanye Ma, Mark Goldstein, Michael S. Albergo, Nicholas M. Boffi, Eric Vanden-Eijnden, and Saining Xie. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers, 2024. URL https://arxiv.org/abs/2401.08740.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Charlie Nash, Jacob Menick, Sander Dieleman, and Peter W Battaglia. Generating images with sparse representations. *arXiv preprint arXiv:2103.03841*, 2021.

Georg Ostrovski, Will Dabney, and Rémi Munos. Autoregressive quantile networks for generative modeling. In *International conference on machine learning*, pp. 3936–3945. PMLR, 2018.

William Peebles and Saining Xie. Scalable diffusion models with transformers, 2023. URL https://arxiv.org/abs/2212.09748.

Yansong Peng, Kai Zhu, Yu Liu, Pingyu Wu, Hebei Li, Xiaoyan Sun, and Feng Wu. Facm: Flow-anchored consistency models, 2025. URL https://arxiv.org/abs/2507.03738.

Ed Pizzi, Sreya Dutta Roy, Sugosh Nagavara Ravindra, Priya Goyal, and Matthijs Douze. A self-supervised descriptor for image copy detection. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14512–14522. IEEE, 2022.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Ali Razavi, Aäron van den Oord, Ben Poole, and Oriol Vinyals. Preventing posterior collapse with delta-vaes. *arXiv preprint arXiv:1901.03416*, 2019.

Jie Ren, Peter J Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. Likelihood ratios for out-of-distribution detection. *Advances in neural information processing systems*, 32, 2019.

Sucheng Ren, Qihang Yu, Ju He, Xiaohui Shen, Alan Yuille, and Liang-Chieh Chen. Flowar: Scale-wise autoregressive image generation meets flow matching, 2024. URL https://arxiv.org/abs/2412.15205.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pp. 1530–1538. PMLR, 2015.

Danilo Jimenez Rezende and Fabio Viola. Taming vaes. *arXiv preprint arXiv:1810.00597*, 2018.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pp. 1278–1286. PMLR, 2014.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. URL https://arxiv.org/abs/2112.10752.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

Axel Sauer, Katja Schwarz, and Andreas Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets, 2022. URL https://arxiv.org/abs/2202.00273.

Axel Sauer, Tero Karras, Samuli Laine, Andreas Geiger, and Timo Aila. Stylegan-t: Unlocking the power of gans for fast large-scale text-to-image synthesis. In *International conference on machine learning*, pp. 30105–30118. PMLR, 2023.

Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9240–9249. IEEE, 2020.

Gianluigi Silvestri and Luca Ambrogioni. Covae: Consistency training of variational autoencoders, 2025. URL https://arxiv.org/abs/2507.09103.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. pmlr, 2015.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020a.

Yang Song and Prafulla Dhariwal. Improved techniques for training consistency models, 2023. URL https://arxiv.org/abs/2310.14189.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020b.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023.

Peize Sun, Yi Jiang, Shoufa Chen, Shilong Zhang, Bingyue Peng, Ping Luo, and Zehuan Yuan. Autoregressive model beats diffusion: Llama for scalable image generation. *arXiv preprint arXiv:2406.06525*, 2024.

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction, 2024. URL https://arxiv.org/abs/2404.02905.

Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*, 2017.

Jakub Tomczak and Max Welling. Vae with a vampprior. In *International conference on artificial intelligence and statistics*, pp. 1214–1223. PMLR, 2018.

Shangyuan Tong, Nanye Ma, Saining Xie, and Tommi Jaakkola. Flow map distillation without data, 2025. URL https://arxiv.org/abs/2511.19428.

Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder, 2021. URL https://arxiv.org/abs/2007.03898.

Arash Vahdat, Karsten Kreis, and Jan Kautz. Score-based generative modeling in latent space, 2021. URL https://arxiv.org/abs/2106.05931.

Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International conference on machine learning*, pp. 1747–1756. PMLR, 2016.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Thuy Phuong Vu, Mai Viet Hoang Do, Minhhuy Le, Dinh-Cuong Hoang, and Phan Xuan Tan. Memorization control in diffusion models from denoising-centric perspective. *arXiv preprint arXiv:2601.21348*, 2026.

Chaoyang Wang and Yunhai Tong. One-step distillation of discrete diffusion image generators via fixed-point iteration, 2026. URL https://arxiv.org/abs/2605.21484.

Shuai Wang, Ziteng Gao, Chenhui Zhu, Weilin Huang, and Limin Wang. Pixnerd: Pixel neural field diffusion, 2025a. URL https://arxiv.org/abs/2507.23268.

Shuai Wang, Zhi Tian, Weilin Huang, and Limin Wang. Ddt: Decoupled diffusion transformer, 2025b. URL https://arxiv.org/abs/2504.05741.

Zidong Wang, Yiyuan Zhang, Xiaoyu Yue, Xiangyu Yue, Yangguang Li, Wanli Ouyang, and Lei Bai. Transition models: Rethinking the generative learning objective, 2025c. URL https://arxiv.org/abs/2509.04394.

Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16133–16142. Ieee, 2023.

Ge Wu, Shen Zhang, Ruijing Shi, Shanghua Gao, Zhenyuan Chen, Lei Wang, Zhaowei Chen, Hongcheng Gao, Yao Tang, Jian Yang, Ming-Ming Cheng, and Xiang Li. Representation entanglement for generation: Training diffusion transformers is much easier than you think, 2025. URL https://arxiv.org/abs/2507.01467.

Weihao Xia, Yulun Zhang, Yujiu Yang, Jing-Hao Xue, Bolei Zhou, and Ming-Hsuan Yang. Gan inversion: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 45(3): 3121–3138, 2022.

Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion gans. *arXiv preprint arXiv:2112.07804*, 2021.

Haowen Xu, Wenxiao Chen, Jinlin Lai, Zhihan Li, Youjian Zhao, and Dan Pei. On the necessity and effectiveness of learning the prior of variational auto-encoder. *arXiv preprint arXiv:1905.13452*, 2019.

Yilun Xu, Ziming Liu, Max Tegmark, and Tommi Jaakkola. Poisson flow generative models, 2022. URL https://arxiv.org/abs/2209.11178.

Yilun Xu, Ziming Liu, Yonglong Tian, Shangyuan Tong, Max Tegmark, and Tommi Jaakkola. Pfgm++: Unlocking the potential of physics-inspired generative models, 2023. URL https://arxiv.org/abs/2302.04265.

Jiawei Yang, Zhengyang Geng, Xuan Ju, Yonglong Tian, and Yue Wang. Representation fr\'echet loss for visual generation. *arXiv preprint arXiv:2604.28190*, 2026a.

Jiawei Yang, Tianhong Li, Lijie Fan, Yonglong Tian, and Yue Wang. Latent denoising makes good tokenizers, 2026b. URL https://arxiv.org/abs/2507.15856.

Jingfeng Yao, Bin Yang, and Xinggang Wang. Reconstruction vs. generation: Taming optimization dilemma in latent diffusion models, 2025. URL https://arxiv.org/abs/2501.01423.

Zeqi Ye, Qijie Zhu, Molei Tao, and Minshuo Chen. Provable separations between memorization and generalization in diffusion models. In *International Conference on Learning Representations*, volume 2026, pp. 143375–143425, 2026.

Qihang Yu, Qihao Liu, Ju He, Xinyang Zhang, Yang Liu, Liang-Chieh Chen, and Xi Chen. Autoregressive image generation with masked bit modeling, 2026a. URL https://arxiv.org/abs/2602.09024.

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, 2025. URL https://arxiv.org/abs/2410.06940.

Yongsheng Yu, Wei Xiong, Weili Nie, Yichen Sheng, Shiqiu Liu, and Jiebo Luo. Pixeldit: Pixel diffusion transformers for image generation, 2026b. URL https://arxiv.org/abs/2511.20645.

Kaiyu Yue, Menglin Jia, Ji Hou, and Tom Goldstein. Image generation with a sphere encoder. *arXiv preprint arXiv:2602.15030*, 2026.

Huijie Zhang, Aliaksandr Siarohin, Willi Menapace, Michael Vasilkovsky, Sergey Tulyakov, Qing Qu, and Ivan Skorokhodov. Alphaflow: Understanding and improving meanflow models, 2025. URL https://arxiv.org/abs/2510.20771.

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

Boyang Zheng, Nanye Ma, Shengbang Tong, and Saining Xie. Diffusion transformers with representation autoencoders, 2025a. URL https://arxiv.org/abs/2510.11690.

Kaiwen Zheng, Yongxin Chen, Huayu Chen, Guande He, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Direct discriminative optimization: Your likelihood-based visual generative model is secretly a gan discriminator, 2025b. URL https://arxiv.org/abs/2503.01103.

Linqi Zhou, Stefano Ermon, and Jiaming Song. Inductive moment matching, 2025. URL https://arxiv.org/abs/2503.07565.

Mingyuan Zhou, Huangjie Zheng, Yi Gu, Zhendong Wang, and Hai Huang. Adversarial score identity distillation: Rapidly surpassing the teacher in one step, 2024. URL https://arxiv.org/abs/2410.14919.

Yuanzhi Zhu, Xi Wang, Stéphane Lathuilière, and Vicky Kalogeiton. Di[M]o: Distilling masked diffusion models into one-step generator, 2025. URL https://arxiv.org/abs/2503.15457.

## APPENDIX CONTENTS


**A Related Work** **20**

A.1 Generative Models Related to G-LZN . . . . . 20

A.2 Addressing Limitations of Existing Generative Models . . . . . 21

A.2.1 Post-Hoc Improvements to Existing Generative Models . . . . . 21

A.2.2 Combining Multiple Generative Paradigms . . . . . 22

**B Proof** **23**

B.1 Proof of Thm. 1 . . . . . 23

B.2 Proof of Thm. 2 . . . . . 24

B.3 Proof of Corollary 1 . . . . . 24

**C Method Details** **25**

C.1 Reconstruction Loss . . . . . 25

C.1.1 Sample-wise Improved Perceptual Loss . . . . . 25

C.1.2 Group-wise Fréchet Loss . . . . . 25

C.2 Architecture . . . . . 27

C.3 Pseudocode . . . . . 28

**D Experimental Details of the Toy Example** **29**

**E Experimental Details of the Main Experiments** **29**

**F Additional Experiments and Analysis** **30**

F.1 Generated Images . . . . . 30

F.2 Image Editing without Task-Specific Training . . . . . 33

F.2.1 Changing Image Classes . . . . . 33

F.2.2 Class Interpolation . . . . . 36

F.2.3 Latent Interpolation . . . . . 36

F.2.4 Editing Image Attributes in Latent Space . . . . . 39

F.2.5 Image Super-Resolution and Inpainting . . . . . 41

F.3 Image Reconstruction . . . . . 41

F.4 Latent Distributions . . . . . 41

F.5 Evidence of Generalization . . . . . 42

F.5.1 Qualitative Results . . . . . 42

F.5.2 Quantitative Results . . . . . 42

F.6 Controlling Memorization and Generalization via Latent Zone Size . . . . . 46

F.7 Stable Training . . . . . 47

F.8 Comparison with Variational Autoencoders . . . . . 47

F.9 Ablation Studies and Hyperparameter Analysis . . . . . 49

F.9.1 Sample-wise Reconstruction Loss . . . . . 49
F.9.2 Group-wise Reconstruction Loss . . . . . 53
F.9.3 Model Size . . . . . 54
F.9.4 Whether to Use a Pretrained Autoencoder . . . . . 54
F.9.5 Zoning Layers at a Fixed Latent Zone Size . . . . . 55
F.9.6 Number of Anchors at a Fixed Latent Zone Size . . . . . 55
F.9.7 Latent Dimension . . . . . 56
F.10 Extended Image Generation Benchmarks . . . . . 56

# A Related Work

In this section, we discuss related work from two aspects (1) the generative models that are related to G-LZNs, and (2) approaches that address the limitations of existing generative models.

## A.1 Generative Models Related to G-LZN

The encoder-decoder framework underlying G-LZNs (§ 3.1) is widely used in generative modeling. However, different designs of the encoding and decoding processes lead to fundamentally different training procedures and properties. Below, we discuss two families of methods most closely related to G-LZNs.

**Diffusion models.** Diffusion models (DMs) (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song & Ermon, 2019) can also be viewed as an encoder-decoder framework with a Gaussian latent space. The mapping between latent and data spaces can be stochastic under the SDE formulation (Song et al., 2020b) or deterministic under the ODE formulation (Song et al., 2020b;a). Unlike G-LZNs, however, this mapping is prescribed by a predefined noising or interpolation process rather than freely learned. This leads to two key limitations: **(1) no native one-step generation**, due to iterative denoising or ODE/SDE integration; and **(2) limited latent flexibility**, as the latent and data spaces must have the same dimensionality and their mapping is constrained by the diffusion formulation.

A popular alternative is to apply diffusion in the latent space of a pretrained autoencoder (Rombach et al., 2022). However, this does not eliminate the intrinsic constraints of diffusion: its latent space remains constrained relative to the autoencoder-induced "data space," and generation still requires multiple steps.

The closest diffusion-based approaches are methods that jointly train the autoencoder and latent diffusion model, such as LSGMs (Vahdat et al., 2021). Joint training allows the latent representation to adapt to the generative objective and provides greater flexibility in both latent dimensionality and latent mapping. However, the resulting formulation remains fundamentally different from G-LZNs in two respects. First, these methods explicitly train a diffusion model in the latent space and require multi-step diffusion sampling. In contrast, in G-LZNs, the flow-matching operation used for latent zoning (§ 3.2) is computed analytically and differentiated through during end-to-end training, without introducing a diffusion process during generation, thereby enabling native one-step generation. Second, these approaches retain VAE-style objectives that combine reconstruction with competing regularization terms, whereas G-LZNs require only stable reconstruction losses.

**Variational and other autoencoders.** Variational autoencoders (VAEs) (Rezende et al., 2014; Kingma & Welling, 2013) also adopt an encoder-decoder framework with a Gaussian latent space. In contrast to DMs, their decoders support one-step generation, and both the latent dimensionality and the mapping between latent and data spaces are flexible. However, the VAE training objective contains competing terms. The reconstruction loss encourages each sample's posterior to be informative and concentrated, whereas KL regularization pushes it toward the uninformative standard Gaussian prior. Balancing these objectives can be challenging. If the KL regularization is too weak, "holes" may remain in the latent space where the model receives little training signal, leading to degraded generated samples (Bauer & Mnih, 2019; Tomczak & Welling, 2018; Xu et al., 2019; Rezende & Viola, 2018; Hoffman et al., 2016). If it is too strong, posterior collapse may occur,

where the posterior approaches the prior and the latent representation becomes uninformative (He et al., 2019; Bowman et al., 2016; Razavi et al., 2019). We also reproduce such training instability in our experiments, including frequent loss and gradient spikes (§ F.8). G-LZNs avoid both issues: latent zoning can cover the prior distribution without holes (§ 3.3), while requiring no competing regularization terms.

One line of follow-up work replaces KL regularization with alternative mechanisms for matching the aggregated posterior to the prior, such as adversarial or MMD losses (Tolstikhin et al., 2017; Makhzani et al., 2015). However, these methods retain the inherent competition between reconstruction and latent regularization, which G-LZNs avoid by construction.

Other work relaxes the requirement that the aggregated posterior follow a simple prior distribution (Ghosh et al., 2019; Van Den Oord et al., 2017; Bojanowski et al., 2017). These approaches still have several limitations. First, some retain weaker forms of competing objectives. For example, in Ghosh et al. (2019), reconstruction encourages the encodings of different samples to separate, whereas latent-norm regularization pulls them together. Second, they sacrifice some desirable properties of VAEs, such as likelihood estimation and, for some methods (Bojanowski et al., 2017), native inversion. Third, and most importantly, because the aggregated posterior is no longer a simple known distribution, these methods require a separate second-stage process to fit a prior for generation. In contrast, G-LZNs ensure a Gaussian aggregated posterior without competing objectives when $S = 1/n$ (§ 3.3). One may argue that when $S < 1/n$, the aggregated posterior is also no longer exactly Gaussian. Nevertheless, G-LZNs retain two stronger structural guarantees: each sample-specific posterior is exactly the standard Gaussian prior restricted to its latent zone, and the zones of different samples have equal probability mass under the Gaussian prior. Thus, the constituent posteriors remain explicitly structured relative to the same known prior, which may explain why G-LZNs work well empirically by sampling directly from the Gaussian prior without learning a separate prior model.

The most closely related work in this space is the concurrent Sphere Encoder (Yue et al., 2026). It maps images to a high-dimensional sphere, and the reconstruction loss encourages the aggregated latent distribution to approach the uniform distribution on the sphere. However, the two approaches differ in several important aspects. **(1)** Besides reconstruction loss, Sphere Encoder uses additional latent- and pixel-consistency losses involving surrogate targets and stop-gradient operations, whereas G-LZNs require only reconstruction losses for stable training. **(2)** Its sample-specific posterior is constructed by adding Gaussian noise to a fixed encoder embedding and projecting the result onto the sphere. Such posteriors are relatively constrained and need not cover the entire prior space, potentially leaving holes similar to those in VAEs. In contrast, latent zoning in G-LZNs produces highly flexible and learnable posterior regions that can collectively cover the prior space (Fig. 3). In § F.9.5 and F.9.6, we further show empirically that flexible and learnable posterior mappings contribute substantially to the performance of G-LZNs, which may also help explain G-LZNs' stronger image-generation quality than Sphere Encoder (Tab. 2a).

### A.2 Addressing Limitations of Existing Generative Models

Different generative paradigms have different inherent limitations (§ 2 and Tab. 1). Much prior work addresses these limitations through post-hoc modifications to an existing paradigm (§ A.2.1) or by combining multiple generative models (§ A.2.2). In contrast, our goal is to develop a new generative model whose desirable properties arise by design, without requiring such fixes.

A comprehensive survey is beyond the scope of this work. We discuss only representative examples to illustrate these directions.

#### A.2.1 Post-Hoc Improvements to Existing Generative Models

Here, we focus on prior work addressing prominent limitations of three widely used image-generation paradigms: the generation inefficiency of diffusion and AR models, and the training instability of GANs.

**Improving generation efficiency of diffusion and AR models.** Diffusion and AR models achieve strong image-generation performance (Tabs. 9 and 10), but their iterative denoising and sequential

generation make inference slow and expensive. Many approaches have therefore been proposed to reduce their generation cost.

Distillation is a major direction. For diffusion models, a pretrained multi-step sampler can provide training targets for a faster student model. Early work distills the entire denoising process into a single-step generator (Luhman & Luhman, 2021), while progressive distillation repeatedly distills multiple sampling steps into fewer steps, halving the required number of steps at each stage (Salimans & Ho, 2022). Consistency-based methods take a related approach by learning mappings that remain consistent along diffusion or flow trajectories, enabling one- or few-step generation (Song et al., 2023; Kim et al., 2024; Geng et al., 2024; Song & Dhariwal, 2023; Peng et al., 2025).

Accelerating AR models through distillation is less straightforward because their output distribution is defined through sequential conditional distributions rather than a direct mapping from noise to samples. Distilled Decoding 1 (Liu et al., 2025b) addresses this by first using flow matching to construct a deterministic mapping from Gaussian noise to the output distribution of a pretrained AR model and then distilling this mapping into one-step generation. Distilled Decoding 2 (Liu et al., 2025a) instead avoids prescribing a particular noise-to-data mapping: it uses the conditional distributions provided by the AR model to construct conditional scores and trains a one-step generator through conditional score distillation.

Other approaches improve generation efficiency without directly distilling the entire generative process. For diffusion models, rectified-flow methods learn straighter transport trajectories that can be accurately integrated with fewer steps (Liu et al., 2022; Lee et al., 2024); different denoisers can be assigned to different stages of the generation process (Liu et al., 2023; Balaji et al., 2022); and saved checkpoints can be linearly combined to improve the quality–efficiency trade-off of diffusion and consistency models (Liu et al., 2025c). For discrete diffusion models, generation can also be accelerated by optimizing which tokens are sampled at each iteration (Liu et al., 2026a;b). Related techniques for AR models combine, select, or prune model components such as experts to reduce inference cost while retaining model quality (Liu et al., 2024).

**Addressing training instability of GANs.** Before the rise of diffusion models, GANs were a dominant paradigm for high-quality image generation. Their training jointly optimizes a generator and discriminator with opposing objectives, which can lead to unstable optimization, mode collapse, and sensitivity to training choices (Goodfellow, 2016). These challenges motivated extensive work on improving GAN training.

One direction modifies the training objective. Wasserstein GANs replace the original adversarial divergence with the Wasserstein distance (Arjovsky & Bottou, 2017; Arjovsky et al., 2017; Gulrajani et al., 2017). Other methods directly address mode collapse; for example, PacGAN allows the discriminator to jointly inspect multiple samples, making collapsed distributions easier to distinguish (Lin et al., 2018). A second direction regularizes or normalizes the networks, such as through spectral normalization of the discriminator (Miyato et al., 2018; Lin et al., 2021). A third direction develops architectural and training advances that improve image quality and practical training behavior, including DCGAN (Radford et al., 2015), progressive growing (Karras et al., 2017), StyleGAN (Karras et al., 2019), and StyleGAN2 (Karras et al., 2020b).

These methods substantially improve the efficiency or stability of established generative paradigms, but do so by modifying, augmenting, or optimizing the underlying paradigm. In contrast, G-LZNs take a different approach: we design a new generative model whose desirable properties arise directly from its formulation rather than being recovered through post-hoc fixes.

### A.2.2 Combining Multiple Generative Paradigms

Different generative paradigms have complementary strengths and weaknesses (§ 2 and Tab. 1). Another line of research therefore combines multiple paradigms within a single framework, aiming to retain their complementary advantages while mitigating their individual limitations. Such combinations can be trained either jointly in a single stage or sequentially in multiple stages.

**Joint or end-to-end combinations.** Denoising Diffusion GANs (Xiao et al., 2021) combine diffusion models with GANs by modeling each denoising transition with a conditional GAN. The expressive adversarial model allows larger denoising steps than the Gaussian transitions typically used in diffusion models, substantially reducing the number of sampling steps while maintaining

strong sample quality and mode coverage. Adversarial Autoencoders (Makhzani et al., 2015) combine autoencoder reconstruction with adversarial training in the latent space: a discriminator encourages the aggregated posterior produced by the encoder to match a chosen prior distribution. Wasserstein Autoencoders (Tolstikhin et al., 2017) provide a broader formulation based on optimal transport, similarly regularizing the aggregated posterior toward the prior through adversarial or other distribution-matching objectives. Latent Score-based Generative Models (Vahdat et al., 2021) instead combine VAEs with score-based generative models, jointly training a VAE and a score model that learns the distribution in its latent space.

**Multi-stage combinations.** Other methods separate representation learning and generative modeling into multiple stages. VQ-VAE (Van Den Oord et al., 2017) first learns a discrete latent representation using a vector-quantized autoencoder and then trains an autoregressive prior over the resulting latent codes. VQGAN (Esser et al., 2021) similarly learns a vector-quantized autoencoder, using perceptual and adversarial objectives to obtain high-quality reconstructions, and subsequently trains an autoregressive Transformer over the learned discrete codes. Latent diffusion models (Rombach et al., 2022) follow a related strategy with continuous latents: they first train an autoencoder to obtain a lower-dimensional representation and then train a diffusion model in this latent space, substantially reducing the computational cost relative to pixel-space diffusion.

These approaches can inherit complementary strengths from multiple generative paradigms, but they can also retain some of their constituent limitations or introduce additional complexity. For example, latent diffusion models still require iterative denoising, VQ-VAE and VQGAN retain autoregressive generation in their learned latent spaces, and methods incorporating GAN objectives still require adversarial training. They may also require multiple models, objectives, or training stages. G-LZNs take a different approach: rather than combining existing generative paradigms to compensate for their respective limitations, we design a new generative model from scratch whose desirable properties arise directly from its formulation.

# B PROOF

## B.1 PROOF OF THM. 1

*Proof.* Let $P_Z = \mathcal{N}(0, \mathbf{I})$, and denote the latent zone associated with $x_i$ by $\mathcal{Z}_i$. By the construction in § 3.2.3, for every measurable set $A$,

$$P_Z(\mathcal{Z}_i) = {w_1}^l \quad \text{and} \quad P_{Z|x_i}(A) = \frac{P_Z(A \cap \mathcal{Z}_i)}{P_Z(\mathcal{Z}_i)}. \tag{5}$$

The reconstruction loss is an average of nonnegative terms. Assumption (iii) therefore implies, for every $i$,

$$\mathbb{E}_{Z \sim P_{Z|x_i}} \left( \mathbb{E}_{\widehat{X} \sim P_D(\cdot|Z)} \left( d\left(x_i, \widehat{X}\right) \right) \right) = 0. \tag{6}$$

Because the distance is nonnegative, it can have zero expectation only if it is zero almost surely. Assumption (i), together with Eq. (5), then gives

$$P_D(\{x_i\} \mid z) = 1 \quad \text{for } P_Z\text{-almost every } z \in \mathcal{Z}_i. \tag{7}$$

We first show that the zones are pairwise disjoint up to $P_Z$-null sets. If $P_Z(\mathcal{Z}_i \cap \mathcal{Z}_j) > 0$ for some $i \neq j$, then Eq. (7) would require the same conditional distribution to assign probability one to both $x_i$ and $x_j$ on a subset of positive $P_Z$-measure. This contradicts assumption (ii). Hence $P_Z(\mathcal{Z}_i \cap \mathcal{Z}_j) = 0$ whenever $i \neq j$.

By assumption (iv), each zone has probability $1/n$. Their pairwise almost-disjointness thus yields

$$P_Z\left( \bigcup_{i=1}^{n} \mathcal{Z}_i \right) = \sum_{i=1}^{n} P_Z(\mathcal{Z}_i) = 1. \tag{8}$$

Consequently, for every measurable set $B$ in the data space, the generated distribution satisfies

$$P_{\widehat{X}}(B) = \int P_D(B \mid z) \mathrm{d}P_Z(z) \tag{9}$$

$$= \sum_{i=1}^{n} \int_{\mathcal{Z}_i} P_D(B \mid z) \mathrm{d}P_Z(z) \tag{10}$$

$$= \sum_{i=1}^{n} P_Z(\mathcal{Z}_i)\, \mathbb{1}[x_i \in B] \tag{11}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}[x_i \in B]. \tag{12}$$

This is precisely the empirical distribution of the training data. □

### B.2 Proof of Thm. 2

*Proof.* Fix any sample $x$, denote its associated latent zone by $\mathcal{Z}_x$, and let $p_Z$ be the density of $P_Z$. Set $m = P_Z(\mathcal{Z}_x) = {w_1}^l$. The conditional latent density for $x$ is

$$q_x(z) \triangleq p(z \mid x) = \frac{p_Z(z)\mathbb{1}[z \in \mathcal{Z}_x]}{m}. \tag{13}$$

It follows immediately that

$$\mathrm{KL}\left(q_x \,\|\, p_Z\right) = \int_{\mathcal{Z}_x} q_x(z) \log \frac{q_x(z)}{p_Z(z)} \mathrm{d}z \tag{14}$$

$$= -\log m = -l \log w_1. \tag{15}$$

Using the nonnegativity of the integrand and then Jensen's inequality gives

$$\log p_D(x) = \log \int p_D(x \mid z) p_Z(z) \mathrm{d}z \tag{16}$$

$$\geq \log \int_{\mathcal{Z}_x} p_D(x \mid z) p_Z(z) \mathrm{d}z \tag{17}$$

$$= \log \mathbb{E}_{Z \sim q_x} \left( \frac{p_D(x \mid Z) p_Z(Z)}{q_x(Z)} \right) \tag{18}$$

$$\geq \mathbb{E}_{Z \sim q_x} \left( \log p_D(x \mid Z) + \log p_Z(Z) - \log q_x(Z) \right) \tag{19}$$

$$= \mathbb{E}_{Z \sim q_x} \left( \log p_D(x \mid Z) \right) - \mathrm{KL}\left(q_x \,\|\, p_Z\right) \tag{20}$$

$$= \mathbb{E}_{Z \sim P_{Z|x}} \left( \log p_D(x \mid Z) \right) + l \log w_1. \tag{21}$$

□

### B.3 Proof of Corollary 1

*Proof.* The likelihood bound in Thm. 2 holds for $P_{\mathrm{data}}$-almost every sample. Taking its expectation over $X \sim P_{\mathrm{data}}$ gives

$$\mathbb{E}_{X \sim P_{\mathrm{data}}} \left( \log p_D(X) \right) \geq \mathbb{E}_{X \sim P_{\mathrm{data}},\, Z \sim P_{Z|X}} \left( \log p_D(X \mid Z) \right) + l \log w_1. \tag{22}$$

By Eq. (3),

$$\log p_D(X \mid Z) = c(X) - \beta\, \mathbb{E}_{\widehat{X} \sim P_D(\cdot|Z)} \left( d\left(X, \widehat{X}\right) \right).$$

Therefore,

$$\mathbb{E}_{X \sim P_{\mathrm{data}},\, Z \sim P_{Z|X}} \left( \log p_D(X \mid Z) \right) = \mathbb{E}_{X \sim P_{\mathrm{data}}} \left( c(X) \right) - \beta \mathcal{L}_{\mathrm{rec}}, \tag{23}$$

which proves Eq. (4). Since $P_{\text{data}}$, $c$, $\beta$, $w_1$, and $l$ are fixed during optimization, all terms in the bound except $\mathcal{L}_{\text{rec}}$ are constant with respect to the model parameters. □

# C Method Details

## C.1 Reconstruction Loss

### C.1.1 Sample-wise Improved Perceptual Loss

Given a fixed, pretrained embedding model, let $\Phi(x) = \{\phi_j(x)\}_{j\in[J]}$ denote the features extracted from an image $x$. A perceptual distance such as LPIPS (Zhang et al., 2018) takes the general form

$$d_\Phi(x, y) = \sum_{j=1}^{J} \frac{1}{H_j W_j} \sum_{h,w} \left\| \bar{\phi}_j(x)_{:,h,w} - \bar{\phi}_j(y)_{:,h,w} \right\|_p^p, \tag{24}$$

where $\bar{\phi}_j$ denotes channel-normalized features, $H_j$ and $W_j$ denote the heights and widths of the features, and $p = 1$ or 2 gives an MAE or MSE variant.[7] The standard LPIPS feature set $\Phi$ consists only of intermediate feature maps from the pretrained network.

Inspired by the richer feature construction used in Drifting Models (Deng et al., 2026), we augment $\Phi$ in the following ways:

- For every feature map, we add global spatial mean and standard deviation for each channel.
- For every feature map, we add local means and standard deviations computed over non-overlapping $2 \times 2$ patches.
- We add the mean squared value of the encoder input for each channel.

We use the resulting augmented feature collection as $\Phi$ in Eq. (24) and directly minimize $d_\Phi(x, D(L(x)))$ for each training sample.

Note that we borrow only the feature construction from Drifting Models, not its drifting objective; our training objective remains fundamentally different. More importantly, G-LZNs also work with simple $\ell_1$ or $\ell_2$ losses directly in pixel space without pretrained embedding models (§ F.9.1), whereas Drifting Models were reported to fail without such embeddings (Deng et al., 2026).

### C.1.2 Group-wise Fréchet Loss

**Fréchet loss (Yang et al., 2026a).** Let $\phi(x) \in \mathbb{R}^d$ be the representation of an image produced by a fixed embedding model. Fréchet distance (FD) approximates the real and generated feature distributions by Gaussians with statistics $(\mu_r, \Sigma_r)$ and $(\mu_g, \Sigma_g)$, respectively, and computes:

$$\text{FD}_\phi\left((\mu_r, \Sigma_r), (\mu_g, \Sigma_g)\right) = \|\mu_r - \mu_g\|_2^2 + \text{Tr}\left(\Sigma_r + \Sigma_g - 2\left(\Sigma_r^{1/2}\Sigma_g\Sigma_r^{1/2}\right)^{1/2}\right). \tag{25}$$

Reliable estimation of $\Sigma_g$ requires substantially more samples than a typical training batch. The queue-based FD loss (Yang et al., 2026a) addresses this problem by maintaining a large queue of recently generated features.[8] At each iteration, features from the current batch are appended to the queue and the oldest features are removed. FD is computed using the full queue, while gradients are propagated only through the current-batch features. This decouples the population size used to estimate FD from the batch size used for optimization. Yang et al. (2026a) show that directly fine-tuning pretrained generative models using this loss improves generated-image quality.

**Group-wise Fréchet loss (ours).** A single Gaussian can poorly approximate a multimodal feature distribution. In particular, matching only the global mean and covariance does not necessarily align the individual modes with their real counterparts. We therefore partition the data into groups and match the real and generated distributions within each group. For $K$ groups, the direct objective

[7] For simplicity, we omit the layer-specific weighting used in LPIPS from Eq. (24).

[8] Besides the queue-based FD loss, Yang et al. (2026a) also proposes a moving-average-based variant. We experimented with this variant in our early studies but found that it performed substantially worse than the queue-based version for our method, so we did not adopt it.

would be

$$\mathrm{GFD}_\phi = \frac{1}{K}\sum_{c=1}^{K} \mathrm{FD}_\phi\left((\mu_{r,c}, \Sigma_{r,c}), (\mu_{g,c}, \Sigma_{g,c})\right). \tag{26}$$

A naive implementation is expensive because every iteration requires estimating $K$ covariance pairs and evaluating $K$ matrix square roots. Moreover, each group may contain too few queued samples to estimate a high-dimensional covariance reliably.

To avoid this cost, we make a shared within-group covariance approximation, where the real groups share $\Sigma_r^{\mathrm{w}}$ and the generated groups share $\Sigma_g^{\mathrm{w}}$:

$$\Sigma_{r,c} = \Sigma_r^{\mathrm{w}}, \qquad \Sigma_{g,c} = \Sigma_g^{\mathrm{w}}, \qquad c \in [K]. \tag{27}$$

Substituting Eq. (27) into Eq. (26) gives

$$\mathrm{GFD}_\phi = \frac{1}{K}\sum_{c=1}^{K} \|\mu_{r,c} - \mu_{g,c}\|_2^2 + \mathrm{Tr}\left(\Sigma_r^{\mathrm{w}} + \Sigma_g^{\mathrm{w}} - 2\left((\Sigma_r^{\mathrm{w}})^{1/2}\Sigma_g^{\mathrm{w}}(\Sigma_r^{\mathrm{w}})^{1/2}\right)^{1/2}\right). \tag{28}$$

We precompute the real group means and real pooled covariance from the training set, and estimate their generated counterparts from the feature queue. Consequently, Eq. (28) requires only one covariance matrix square root per iteration, just like the original FD loss; computing the group means adds only inexpensive accumulation operations. The proposed group-wise loss therefore has essentially the same computational complexity as the original FD loss while better accounting for multimodal data.

*Sharded feature queue.* A large feature queue can consume substantial GPU memory if it is replicated on every worker. We therefore shard a queue of size $N$ across $R$ workers, such that each worker stores only $N/R$ features and their labels. Each worker computes its local feature outer product, group-wise feature sums, and group counts. These sufficient statistics are then summed across workers to recover the statistics of the full logical queue. Gradients are retained for features from the current local batch.

**Loss normalization.** Yang et al. (2026a) propose normalizing the FD loss as

$$\overline{\mathrm{FD}}_\phi = \frac{\mathrm{FD}_\phi}{\mathrm{sg}\left(\mathrm{FD}_\phi\right)}, \tag{29}$$

where $\mathrm{sg}\left(\cdot\right)$ denotes the stop-gradient operator. Their motivation is to normalize losses from different embedding models $\phi$ to the same forward value of 1. We apply the same normalization to $\mathrm{GFD}_\phi$:

$$\overline{\mathrm{GFD}}_\phi = \frac{\mathrm{GFD}_\phi}{\mathrm{sg}\left(\mathrm{GFD}_\phi\right)}. \tag{30}$$

Our motivation, however, is different. Equal loss values do not imply equal influence on optimization, e.g., in terms of gradient magnitude. Instead, we view this normalization as providing *adaptive weighting*: the gradient is scaled by $1/\mathrm{sg}\left(\mathrm{FD}_\phi\right)$ or $1/\mathrm{sg}\left(\mathrm{GFD}_\phi\right)$. Early in training, when generation quality is poor and the FD loss is large, its gradient is downweighted, allowing the sample-wise improved perceptual loss to first establish good sample-level reconstruction (§ C.1.1). As generation improves and the FD loss decreases, its relative weight increases, automatically shifting optimization toward finer-grained group-wise distribution matching.

Note that unlike the trajectory-level stop-gradient targets in CMs/MF, this is a scalar loss normalization and does not define a self-dependent training target.

**Sample source.** The Fréchet loss requires generated samples to populate the queue. The simplest implementation is to use the reconstructed samples $D\left(L\left(x\right)\right)$ from Eq. (1) (or $D\left(L\left(x, c\right), c\right)$ in the conditional case). However, because we use $S \ll 1/n$, reconstruction training covers only a small fraction of the latent space. To encourage the decoder to learn across the entire latent space, we instead populate the queue with $D\left(z\right)$ (or $D\left(z, c\right)$ in the conditional case), where $z \sim \mathcal{N}\left(0, \mathbf{I}\right)$ is freshly sampled from the prior.

**Experimental settings.** For conditional `ImageNet256×256`, we use the group-wise FD loss, with each group defined by its class label. For unconditional `CIFAR10`, we use the original FD loss because no class information or condition is available; this can be viewed as a special case of group-wise FD loss with a single group.

**Notes on the use of embeddings.** One may be concerned that the Fréchet loss shares embeddings with the evaluation metrics. We note that `G-LZNs` trained without any Inception-based loss already achieve FID 5.47–6.80 (Tab. 6), and the group-wise loss on ImageNet improves $\mathrm{FDr}^6$ despite being less directly aligned with the global $\mathrm{FDr}^6$ statistic (§ F.9.2). Nevertheless, metrics computed in embeddings that also appear in training losses should be interpreted with care; we therefore additionally report precision and recall, and compare against FD-loss-tuned baselines (Tab. 10) under the same convention.

### C.2 Architecture

**Base architecture.** We adopt the ResNet-based NCSN++ (U-Net) architecture introduced for score-based generative modeling (Song et al., 2020b), following the implementation used by Rectified Flow (Liu et al., 2022). We use its downsampling and upsampling paths as our encoder and decoder, respectively, while removing the long-range U-Net skip connections between them. The residual connections within individual ResNet blocks are retained. We also remove the diffusion timestep and noise-level embeddings, since our encoder and decoder do not process noisy intermediate states. All model parameters are trained from scratch.

**Anchor computation.** Let $h_s(x, c)$ denote the encoder feature map at stage $s$, where the class condition $c$ is omitted for unconditional models. We extract feature maps from multiple stages of the downsampling network and transform each selected feature map using a separate projection head:

$$P_s(h_s) = \mathrm{MLP}_s\left(\mathrm{Flatten}\left(\sigma\left(\mathrm{Conv}_{1\times1,s}(h_s)\right)\right)\right), \tag{31}$$

where $\sigma$ denotes the activation function. The projection heads have the same output dimensionality, and their outputs are summed to produce the raw encoder output:

$$u(x, c) = \sum_s P_s\big(h_s(x, c)\big) \in \mathbb{R}^{m(d+1)}, \tag{32}$$

where $d$ is the latent dimensionality and $m = l(k-1)$ is the number of nonzero anchors predicted across all latent-zoning layers. The reason for having "$+1$" in "$d+1$" and "$-1$" in "$k-1$" will be explained later.

The projection heads attached to high-resolution feature maps are omitted. Since these heads flatten their spatial feature maps before applying fully connected layers, their computation and parameter counts grow with spatial resolution. Omitting them therefore substantially reduces the encoder cost. This modification removes only the corresponding projection heads; the underlying high-resolution ResNet blocks remain part of the encoder backbone.

**Anchor normalization.** We apply an anchor-wise *soft normalization* to stabilize the magnitudes of the predicted anchors while allowing some flexibity. Specifically, we divide $u(x, c)$ into $m$ vectors $u_{ij} \in \mathbb{R}^{d+1}$, normalize each vector in $d+1$ dimensions, and then discard its last coordinate:

$$a_{ij} = \rho\sqrt{d+1}\,\frac{u_{ij,1:d}}{\|u_{ij}\|_2}, \qquad i \in [l], \quad j = 2, \ldots, k, \tag{33}$$

where $\rho$ is a range hyperparameter. Because the normalization is performed before the additional coordinate is discarded, the resulting anchor satisfies

$$\|a_{ij}\|_2 \le \rho\sqrt{d+1}, \tag{34}$$

but is not constrained to lie on a fixed-radius hypersphere. This bounds the anchor magnitude while preserving flexibility in its norm.

**Algorithm 1:** The latent computation algorithm $L$

**Input:** Sample $x$; encoder $E$; number of anchors $k$; number of latent zoning layers $l$; anchor weights $\{w_j\}_{j\in[k]}$

**Output:** Latent $z = L(x)$

$\{a_{ij}\}_{i\in[l],\, j\in[k]} \leftarrow E(x)$; `// self-latent zoning (§ 3.2.2); in practice` $a_{i1} = \mathbf{0}$ `(Eq. (35))`

Draw $\epsilon_1 \sim \mathcal{N}(0, \mathbf{I})$;

**for** $i = 1, \ldots, l$ **do**

  $\epsilon_{i+1} \leftarrow \mathrm{IFM}(a_{i1}, \epsilon_i; a_{i1}, \ldots, a_{ik})$ with anchor weights $\{w_j\}_{j\in[k]}$[9]; `// recursive and weighted latent zoning (§ 3.2.3)`

**return** $z = \epsilon_{l+1}$;

**Zero anchor.** To save encoder parameters, for each latent-zoning layer, we fix the first anchor to the origin:

$$a_{i1} = \mathbf{0}, \qquad i \in [l]. \tag{35}$$

The encoder therefore predicts only $a_{i2}, \ldots, a_{ik}$. The zero anchor is prepended when constructing the anchor set and does not require an additional network output.

**Encoder condition input.** For class-conditional models, the discrete class label is represented as a one-hot vector and transformed by a small MLP. The resulting class embedding conditions every encoder ResNet block, analogous to timestep conditioning in diffusion models, through channel-wise shifts and/or scales. The class embedding is omitted for unconditional models.

**Decoder.** The decoder consists only of the upsampling path of NCSN++ and does not receive feature maps from the encoder. It starts from a fixed all-zero feature tensor at the lowest spatial resolution. This tensor is shared across samples and is neither sampled nor learned; sample-specific information is supplied entirely through the latent and, for conditional models, the class label.

Specifically, we concatenate the latent $z$ and the one-hot class vector $c$ and transform them jointly using an MLP:

$$e_D(z, c) = \mathrm{MLP}_D\big([z; c]\big). \tag{36}$$

For unconditional models, the label is omitted and the MLP receives only $z$. The resulting embedding $e_D$ is provided as a conditioning vector to every decoder ResNet block, where it modulates the feature channels through shifts and/or scales, analogous to timestep conditioning in diffusion models.

**Additional notes on conditional generation.** The original Latent Zoning Network (LZN) (Lin et al., 2026b) uses a different approach to conditional generation. Rather than providing the condition as a separate input $c$, LZN encodes all information, including conditions such as class labels, into the latent $z$ itself. Specifically, different data modalities, such as images, text, and class labels, are mapped into a shared latent space, where semantically related inputs are organized into corresponding latent zones. Conditional generation can then be performed using only the latent $z$, without an additional condition input. This design is important for the original goal of LZN, which is to unify generative modeling, classification, and representation learning through compositions of modality-specific encoders and decoders in a shared latent space. In contrast, since this work focuses solely on generative modeling, we adopt the standard conditional-generation formulation with separate latent and condition inputs $(z, c)$, supplying $c$ directly to both the encoder and decoder.

### C.3 Pseudocode

Alg. 1 shows the pseudocode for the latent computation algorithm in § 3.2.

[9] The backward integration in $\mathrm{IFM}(\cdot)$ is approximated with Euler discretization; we use 50 Euler steps in all experiments.

## D EXPERIMENTAL DETAILS OF THE TOY EXAMPLE

Tab. 3 summarizes the experimental details of the toy example (Fig. 3).

Table 3: Experimental settings of the toy example.

| **Hyperparameter** | **Value** |
|---|---|
| Distance function $d$ | $\ell_2$ loss |
| Batch size | $9 \times 1024$ (1,024 latents per data point) |
| Number of latent zoning layers $l$ | 2 |
| Number of anchors per latent zoning layer $k$ | 4 |
| Latent zoning anchor weights $w$ | $[\frac{1}{3}, \frac{2}{9}, \frac{2}{9}, \frac{2}{9}]$ |
| Encoder/decoder architecture | Three-hidden-layer MLPs with 256 units per layer and SiLU activations |

## E EXPERIMENTAL DETAILS OF THE MAIN EXPERIMENTS

Tab. 4 summarizes the experimental settings for the main experiments in § 4. The hyperparameter variations considered in the ablation studies (§ F.9) are based on these settings.

Table 4: Experimental settings of the main experiments.

| **Hyperparameter** | `CIFAR10` | `ImageNet` **(pixel space)** | `ImageNet` **(latent space)** |
|---|---|---|---|
| *Latent computation* | | | |
| Latent dimension | | 50 | |
| Number of latent zoning layers $l$ | | 15 | |
| Number of anchors per latent zoning layer $k$ | | 4 | |
| Latent zoning anchor weights $w$ | | $[\frac{1}{61}, \frac{20}{61}, \frac{20}{61}, \frac{20}{61}]$ | |
| Latent zone size $S = {w_1}^l$ | | $61^{-15} \approx 1.7 \times 10^{-27}$ | |
| Number of Euler steps | | 50 | |
| Anchor range $\rho$ (Eq. (33)) | | 3 | |
| *Architecture* | | | |
| Pretrained autoencoder | None | None | SD-VAE |
| Generation space | $32\times32\times3$ pixels | $256\times256\times3$ pixels | $32\times32\times4$ AE latents |
| Condition | None | Class label | Class label |
| Base channels | 192 | 256 | 384 |
| Channel multipliers | $(2, 2, 2, 2)$ | $(2, 2, 2, 2, 2, 2, 2)$ | $(2, 2, 2, 2)$ |
| ResNet blocks per resolution | | 3 | |
| Attention layers | | None | |
| Dropout | 0.1 | 0.05 | 0.05 |
| *Sample-wise improved perceptual loss (§ C.1.1)* | | | |
| Embedding models | | ConvNeXt V2-Base and DINO ResNet-50 | |
| Distance | | $\ell_1$ | |
| Embedding input resolution | | $224\times224$ | |
| Loss coefficient (per embedding model) | | 0.001 | |
| *Fréchet loss (§ C.1.2)* | | | |
| Grouping | Single group | Per-class groups | Single group |
| Embedding models | Inception | Six FDr$^6$ models | Six FDr$^6$ models |
| Queue size | 50,000 | 65,536 | 65,536 |
| Loss coefficient (per embedding model) | | 0.02 | |
| *Training* | | | |
| Batch size | 4,096 | 2,048 | 2,048 |
| Optimizer | | Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$) | |
| Learning rate (encoder and decoder) | | 0.001 | |
| Learning rate warmup | | Linear, over the first 5,000 iterations | |
| EMA decay (encoder and decoder) | | 0.999999 | |
| Gradient norm clipping | | 1.0 | |
| Data augmentation | | Random horizontal flip | |

# F Additional Experiments and Analysis

## F.1 Generated Images

Figs. 4 and 5 shows generated images on `ImageNet256×256` and `CIFAR10`.



Figure 4: Class-conditional generation on `ImageNet256×256`.



Figure 5: Unconditional generation on `CIFAR10`.

### F.2 Image Editing without Task-Specific Training

In this section, we show that G-LZNs can support a series of image editing applications without any task-specific training.

#### F.2.1 Changing Image Classes

G-LZNs supports changing the class of a given image. We encode each image into latent space and, when decoding the resulting latent, replace the class condition with a different class. Figs. 6 and 7 show the results. G-LZNs preserve the overall structure and background of the source image while realistically changing its main object to different classes. We highlight several interesting observations:

- Class transfer can occur across very different categories. For example, Rows 5 and 6 and Column 3 in the two figures show that cars, animals, toys, and mountains in scenic images can be transformed into one another.
- The transformed results remain realistic. For example, Rows 6 and 7 show that not only does the main object change to the intended class, but its reflection in the water is also adapted accordingly.
- The method can create imaginative yet realistic scenes. For example, Row 6 contains animals or a large strawberry floating or walking on water. Although such scenes are unlikely to occur in nature, the placement and interaction of the objects remain visually realistic.




Figure 6: Changing image classes with G-LZNs (Part 1). The first column shows real images, and the second shows their reconstructions by G-LZNs. Subsequent columns use the same latent $z$ as the reconstruction but different class conditions, indicated at the top. G-LZNs preserve the overall structure and background of the source image while realistically changing its main object to different classes. Zoom in for more details.




Figure 7: Changing image classes with G-LZNs (Part 2). The first column shows real images, and the second shows their reconstructions by G-LZNs. Subsequent columns use the same latent $z$ as the reconstruction but different class conditions, indicated at the top. G-LZNs preserve the overall structure and background of the source image while realistically changing its main object to different classes. Zoom in for more details.

### F.2.2 CLASS INTERPOLATION

During training, the network only sees discrete class conditions represented as one-hot vectors. For this application, we interpolate between two classes by linearly interpolating their one-hot class vectors. Figs. 8 and 9 show the results. Although the network is never exposed to such interpolated class conditions during training, it can still generate realistic images that smoothly transition between the two classes. We highlight several interesting observations:

- The interpolated images remain smooth and realistic throughout the transition. Pixel-level changes are gradual, while the perceived semantic class can change relatively sharply, for example when dogs transition into cats in Fig. 8. We view this as desirable behavior for a generative model with high sample quality and good generalization, in contrast to the claim in Yue et al. (2026) that sharp and abrupt pixel changes during interpolation are necessary for a good generative model.
- Interpolation also works between very different classes, such as a scenic landscape and a zebra in Fig. 8.

### F.2.3 LATENT INTERPOLATION

G-LZNs also support interpolation in latent space, producing smooth transitions between images. Figs. 8 and 9 show the results. We highlight several interesting observations:

- The interpolations are smooth in both pixel space and human perception, while consistently producing realistic images.
- Interpolation can span substantial variations in object position, background, color, distance, and object type.

Such direct interpolation in a compact, learned latent space is less natural in standard data-space diffusion and AR models, whose formulations do not provide such a representation by design (Ho et al., 2020; Van Den Oord et al., 2016).




Figure 8: Class and latent interpolation with G-LZNs (Part 1). In each grid, the top-left and bottom-right images are reconstructions of two real images. The intermediate images are generated by linearly interpolating the class input and spherically linearly interpolating the latent input between them. G-LZNs interpolate smoothly along both axes and produce realistic images throughout the grid. Zoom in for more details.

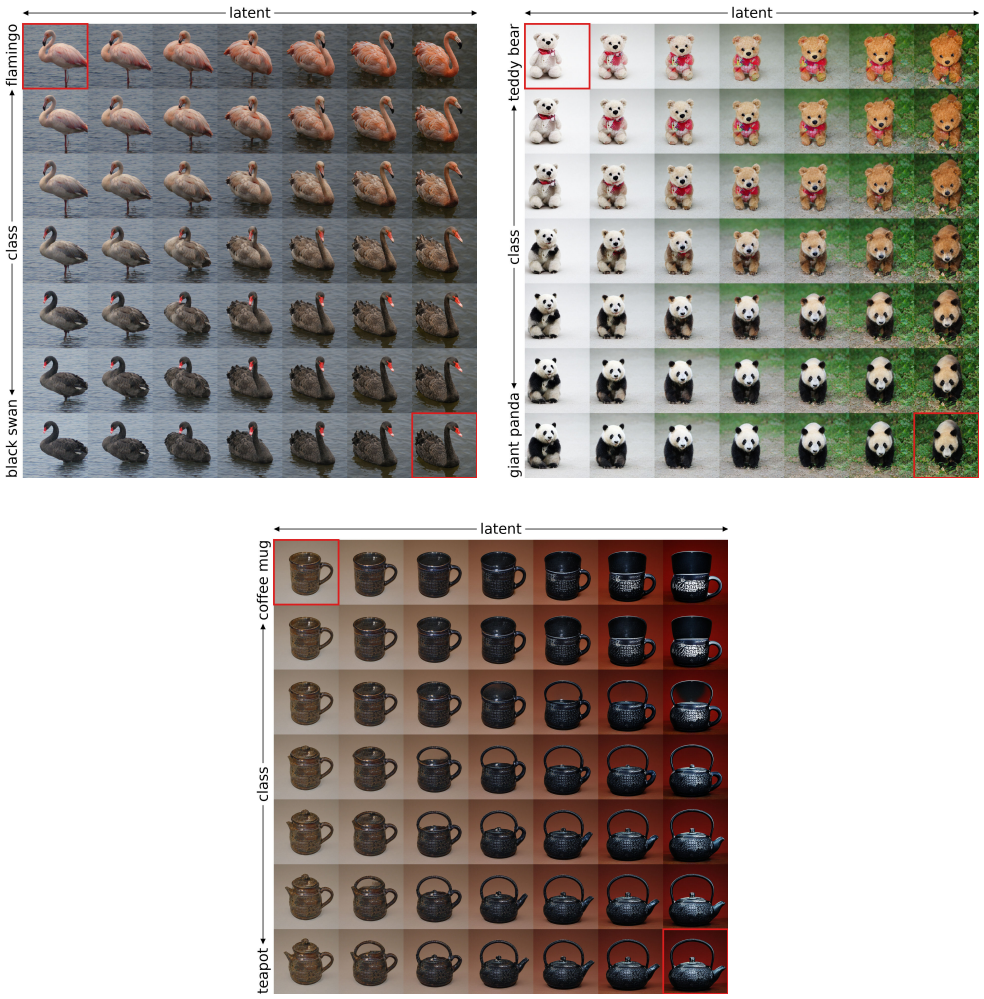


Figure 9: Class and latent interpolation with G-LZNs (Part 2). In each grid, the top-left and bottom-right images are reconstructions of two real images. The intermediate images are generated by linearly interpolating the class input and spherically linearly interpolating the latent input between them. G-LZNs interpolate smoothly along both axes and produce realistic images throughout the grid. Zoom in for more details.

### F.2.4 Editing Image Attributes in Latent Space

Beyond classes, G-LZNs can also modify finer-grained image attributes by simply adding or subtracting directions in latent space. Fig. 10 shows the results. For each target attribute (e.g., mouth opening), we select two groups of representative images (10–16 per group) corresponding to opposite states of the attribute (e.g., mouth open and mouth closed). We compute the average latent of each group and take their difference as the attribute direction $d$. For each query image, we compute its latent $z$ and obtain an edited latent $z' = z + \alpha d$, where $\alpha$, which can be positive or negative, controls the direction and magnitude of the edit. We then rescale $z'$ to preserve the norm of the original latent, $\|z\| \frac{z'}{\|z'\|}$, so that the edited latent has the same density under the isotropic Gaussian prior, and decode it to obtain the edited image. As shown in Fig. 10, this simple procedure can consistently and continuously modify the desired attributes while producing realistic images. These results further demonstrate that the latent space learned by G-LZNs is semantically structured. Such simple linear manipulation of semantic attributes in a compact latent space is less direct in standard data-space diffusion and AR models, whose formulations do not provide such a representation by design (Ho et al., 2020; Van Den Oord et al., 2016).




Figure 10: Changing image attributes through latent manipulation with G-LZNs. In each example, the red box marks the reconstruction of the target image. We then add or subtract latent directions corresponding to the desired attributes (see § F.2.4) to generate edited images. G-LZNs can reliably modify image attributes while preserving realistic appearance, with the magnitude of each change continuously controllable. Zoom in for more details.

### F.2.5 Image Super-Resolution and Inpainting

G-LZNs also support image super-resolution and inpainting without task-specific training. We simply feed the low-resolution image (upsampled to the input resolution) or the masked image into the encoder to obtain a latent, and then decode this latent to produce the restored image. Since G-LZNs learn the manifold of realistic images, this process can be viewed as projecting the corrupted input onto the learned data manifold. Fig. 11 shows the results: the restored images are realistic while preserving the main content of the inputs.




Figure 11: Reconstructions on CIFAR10 for in-distribution (ID) and out-of-distribution (OOD) images. The first row shows the inputs (real ID samples and OOD distortions), and the second row shows their reconstructions. G-LZNs accurately reconstruct ID samples and project OOD samples to realistic images. This ability to restore OOD inputs enables applications such as image super-resolution and inpainting.

## F.3 Image Reconstruction

Figs. 6, 7 and 11 show image reconstructions on CIFAR10 and ImageNet256×256.

## F.4 Latent Distributions

Here, we inspect the latent distribution of training images. Fig. 12 shows that it closely matches the prior distribution (standard Gaussian). Fig. 13 further shows how its mean and standard deviation evolve during training. Although the distribution fluctuates early in training, its mean and standard deviation subsequently remain close to their expected values of 0 and 1. These results suggest that, even with $S \ll 1/n$, the latent distribution of training images can remain empirically close to the Gaussian prior.

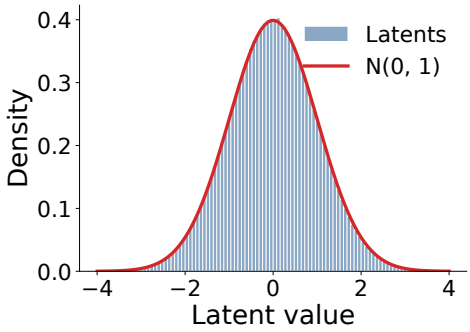


Figure 12: Pooled marginal latent distribution of training images from a trained G-LZN model on CIFAR10. The distribution closely matches the prior distribution (standard Gaussian).

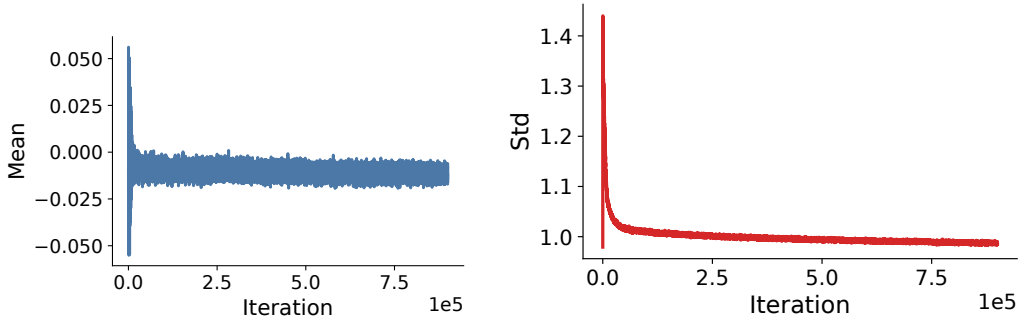


Figure 13: Mean and standard deviation of training-image latents over training iterations on CIFAR10. Except during early training, the mean is close to 0 and the standard deviation is close to 1.

### F.5 Evidence of Generalization

The smooth and realistic latent interpolation and image editing results in § F.2 already suggest that G-LZNs do not simply memorize the training set, but generalize beyond it. In this section, we provide further evidence of this generalization and compare G-LZNs with real validation data and representative baselines.

#### F.5.1 Qualitative Results

We use horizontal-flip data augmentation in all experiments. Here, for randomly generated samples, we retrieve their 10 nearest neighbors from the training set, including horizontally flipped training images. We measure similarity using SSCD embeddings (Pizzi et al., 2022), which are specifically designed for detecting image copying.

Figs. 14 and 15 show the results on CIFAR10 and ImageNet256×256, respectively. The generated samples are visually distinct from even their nearest training examples, providing qualitative evidence that G-LZNs generate novel samples rather than simply memorizing the training set.

#### F.5.2 Quantitative Results

Here, we compare the distribution of SSCD distances to the nearest training images (including horizontally flipped versions) for G-LZNs against two references: (1) the real validation set and (2) strong generative baselines. For CIFAR10, the strongest baseline in terms of FID, PFGM++ (Xu et al., 2023), does not currently provide an accessible checkpoint.[10] We therefore use the second

[10]The GitHub repository provides a checkpoint link, but the linked Google Drive folder is empty at the time of writing (September 2026).

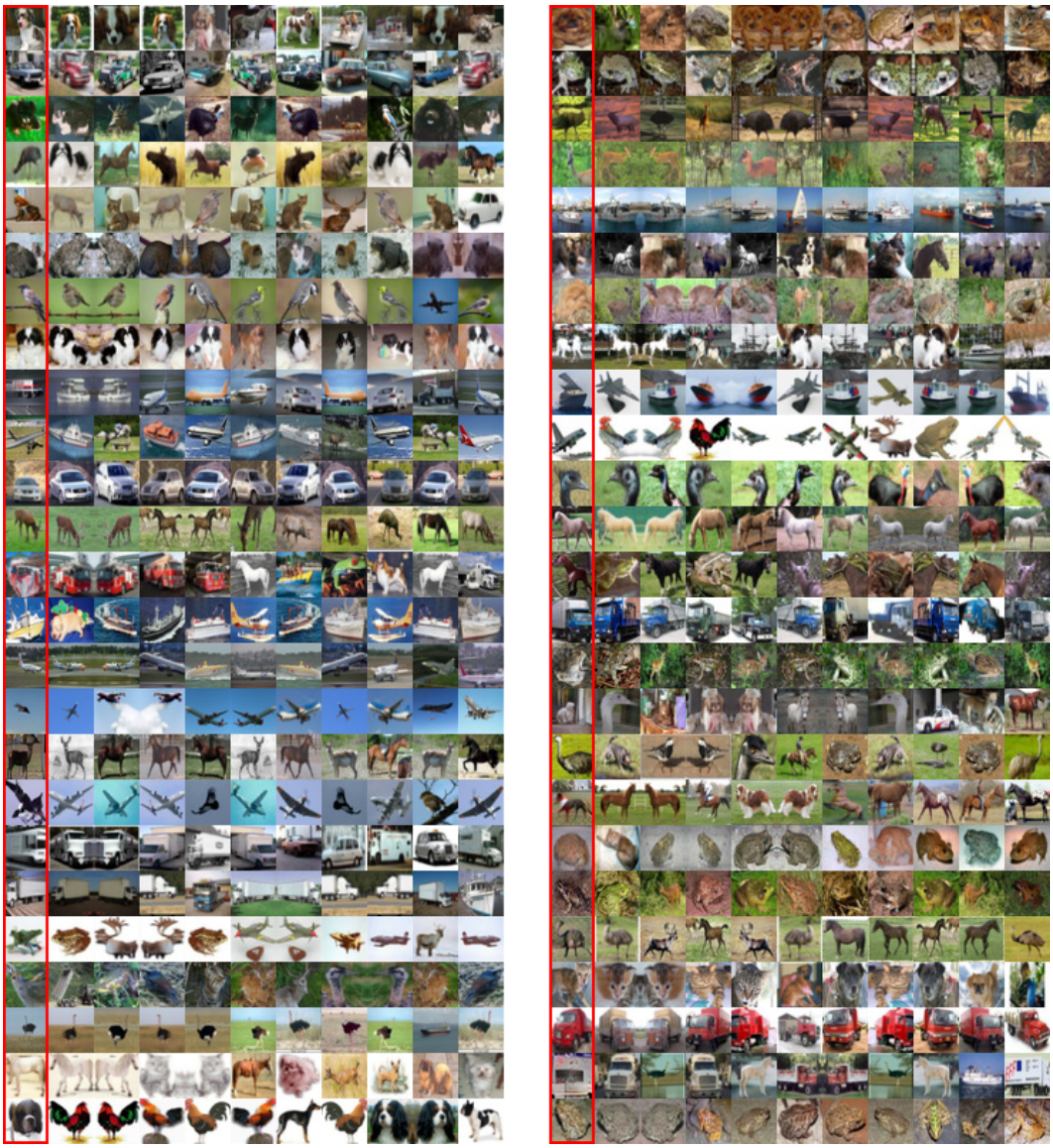

Figure 14: Nearest training images for generated samples on CIFAR10. Red boxes indicate the generated samples, followed by their 10 nearest training images, including horizontally flipped versions. Similarity is measured using SSCD embeddings. G-LZNs generate novel samples that are visually distinct from their nearest training examples.

strongest baseline, EDM (Karras et al., 2022). For ImageNet256×256, we pick W-Flow (Han et al., 2026), the baseline with the best FID score.

Results in Figs. 16 and 17 show that G-LZNs generally have larger distances to the training set than both the real validation set and strong baselines. This suggests that G-LZNs generalize beyond the training set rather than simply memorizing it.

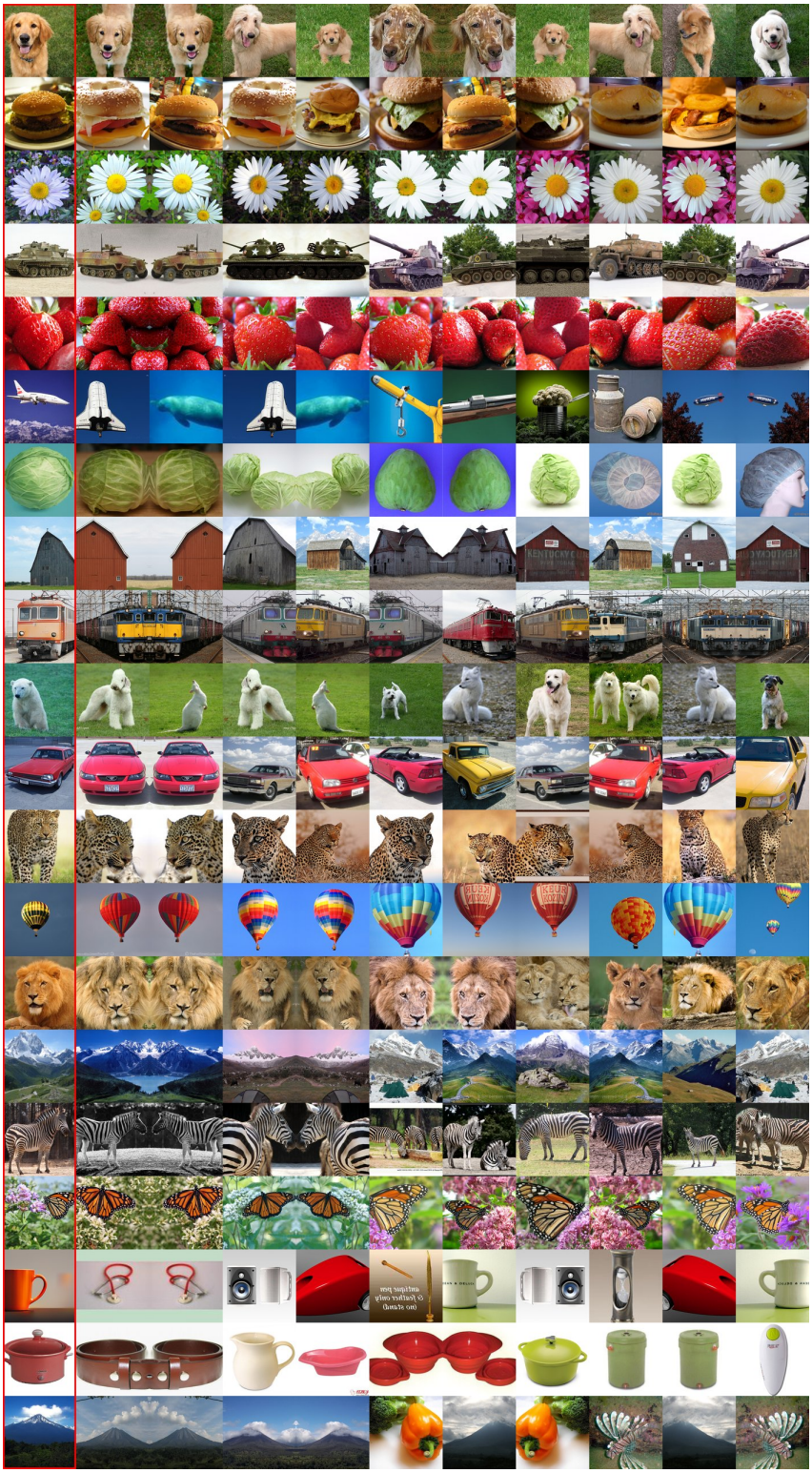

Figure 15: Nearest training images for generated samples on ImageNet256×256. Red boxes indicate the generated samples, followed by their 10 nearest training images, including horizontally flipped versions. Similarity is measured using SSCD embeddings. G-LZNs generate novel samples that are visually distinct from their nearest training examples.

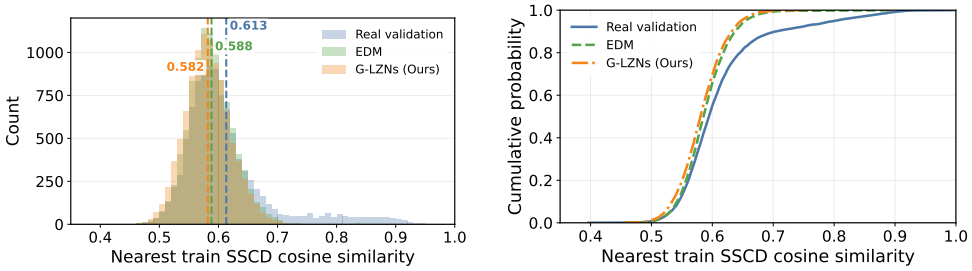


Figure 16: Distribution of the SSCD cosine distance from each generated or validation image to its nearest training image on `CIFAR10`. The left panel shows the histogram, with vertical lines indicating the mean distances. Because the histograms overlap substantially, the right panel additionally shows the corresponding CDFs. `G-LZNs` have larger nearest-training-image distances than both the real validation set and the strong EDM baseline, providing quantitative evidence that `G-LZNs` generalize beyond the training set rather than simply memorizing it.

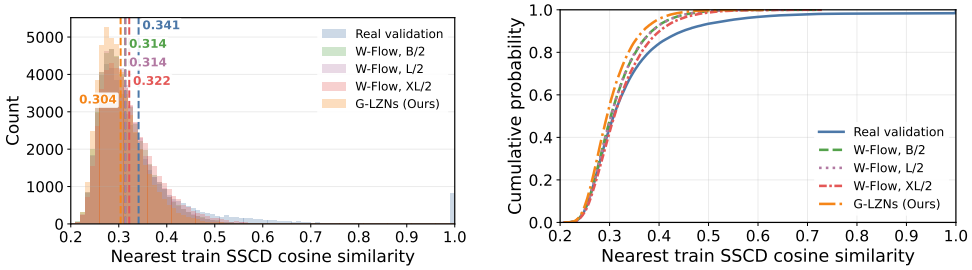


Figure 17: Distribution of the SSCD cosine distance from each generated or validation image to its nearest training image on `ImageNet256×256`. The left panel shows the histogram, with vertical lines indicating the mean distances. Because the histograms overlap substantially, the right panel additionally shows the corresponding CDFs. `G-LZNs` have larger nearest-training-image distances than both the real validation set and the strong W-Flow baselines, providing quantitative evidence that `G-LZNs` generalize beyond the training set rather than simply memorizing it.

### F.6 Controlling Memorization and Generalization via Latent Zone Size

In § F.5, we showed that G-LZNs generalize well beyond the training set. However, different applications may favor different points on the memorization–generalization trade-off, and stronger memorization can be desirable when generated samples are expected to closely reflect the training distribution (Vu et al., 2026; Hartmann et al., 2023). Standard generative model formulations typically do not expose an intrinsic parameter for directly controlling this trade-off; instead, memorization is influenced implicitly by factors such as dataset size, model capacity, and training procedure (Gu et al., 2023). In this section, we show that G-LZNs provide such a direct control through the latent zone size $S$.

We vary the latent zone size over a wide range, from $4 \times 10^{-6}$ to $4 \times 10^{-72}$, by changing the number of latent zoning layers $l$ from 3 to 40. All other settings, including the architecture and hyperparameters, are kept fixed. (For reference, our main experiments use $l = 15$.)

Figs. 18 and 19 show the degree of memorization, measured by the mean SSCD similarity to the nearest training image (introduced in § F.5), together with generation quality measured by FID. Key observations are:

- **Generation quality is robust to latent zone size.** FID remains low ($\leq 2.6$) across this wide range of latent zone sizes, demonstrating that G-LZNs are robust to this hyperparameter.
- **Latent zone size controls the memorization–generalization trade-off.** Different latent zone sizes lead to substantially different memorization behavior, with larger zones producing more memorization. At $S = 4 \times 10^{-6}$, memorization becomes significant: generated samples have substantially higher SSCD similarity to the training set than real validation images. This behavior is expected because the reconstruction objective explicitly encourages the decoder to reproduce training images within their corresponding latent zones. With horizontal-flip augmentation, CIFAR10 contains $10^5$ training images; if the latent zones are disjoint, a zone size of $4 \times 10^{-6}$ means that the training-image zones collectively occupy about 40% of the latent prior. As these zones occupy a larger fraction of the latent space, the model has less freedom to generate novel samples outside the regions directly constrained by reconstruction.

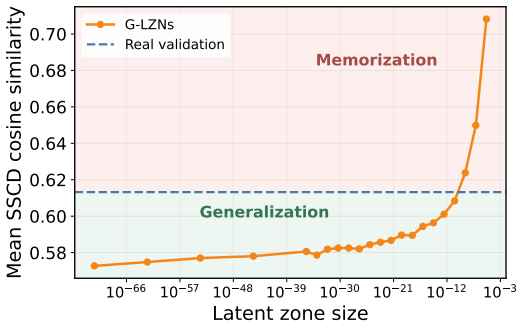


Figure 18: Mean SSCD cosine similarity between each generated image and its nearest training image on CIFAR10 under different latent zone sizes. The horizontal line shows the corresponding mean similarity for validation images as a reference. Latent zone size smoothly controls the degree of memorization, with larger zones leading to stronger memorization.

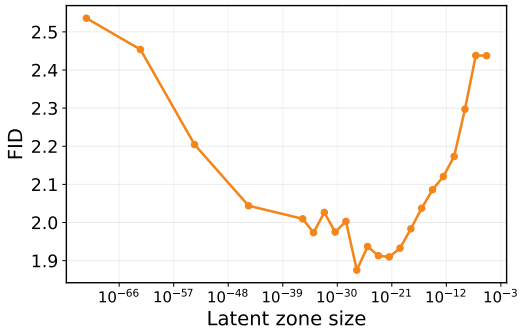


Figure 19: FID on CIFAR10 under different latent zone sizes. FID remains low across a wide range of zone sizes, although extremely small or large zones slightly degrade generation quality.

### F.7 Stable Training

Fig. 20 shows that G-LZN training is stable both within and across runs.

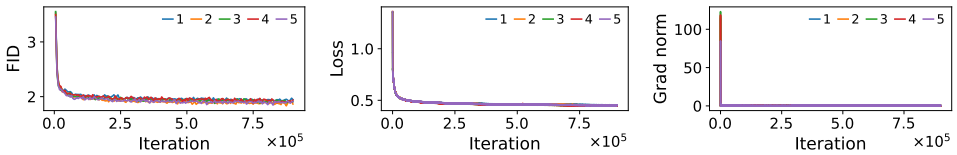


Figure 20: FID, training loss, and gradient norm over training iterations across five random seeds on CIFAR10. G-LZN training is stable within each run and exhibits low variance across runs. The reported loss is the sample-level loss; the group-level loss is constant by design (§ C.1.2). The loss and gradient norm show the *raw values at every iteration without any smoothing*.

### F.8 Comparison with Variational Autoencoders

As discussed in § A.1, G-LZNs are closely related to variational autoencoders (VAEs). VAEs use KL regularization to encourage the aggregated posterior to match a Gaussian prior, but this introduces a competing objective with reconstruction and can lead to training instability. In contrast, G-LZNs use latent zoning to structure the latent distribution toward the same Gaussian prior without introducing a competing loss to maintain stable training (§ 4.3).

In this section, we empirically compare latent zoning in G-LZNs with KL regularization in VAEs, focusing on both training stability and generation quality.

**Experimental setting.** To implement VAEs, we start from the same CIFAR10 experimental setup used for G-LZNs and make only the changes necessary to obtain a VAE: we remove the anchor-computation branches (§ C.2) and introduce the standard KL regularization term. Because our goal is to isolate the effect of latent zoning versus KL regularization, we remove the FD loss (§ C.1.2) from both VAEs and G-LZNs to avoid confounding effects. Under this setting, the VAE reduces to a standard VAE except that it uses the same perceptual reconstruction loss as G-LZNs (§ C.1.1).

To ensure that the VAEs are properly tuned, we perform a fine-grained grid search over the KL regularization weight, ranging from $1 \times 10^{-6}$ to $5 \times 10^{-3}$ with multiplicative steps of 2 or 2.5:

$1\times10^{-6}$, $2\times10^{-6}$, $5\times10^{-6}$, $1\times10^{-5}$, $2\times10^{-5}$, $5\times10^{-5}$, $1\times10^{-4}$, $2\times10^{-4}$, $5\times10^{-4}$, $1\times10^{-3}$, $2\times10^{-3}$, $5\times10^{-3}$.

The smallest and largest weights are chosen to be sufficiently extreme that VAE performance degrades substantially at both ends of the range, as shown later. This helps ensure that the search covers the useful regime of KL regularization strengths.

To ensure sufficient training, we continue each VAE run until the sample quality, measured by FID, has not improved for 50,000 iterations, or until the total number of training iterations reaches 1,000,000.

**G-LZNs have better training stability than VAEs.** Figs. 21 and 22 show the training loss and gradient norms of VAEs and G-LZNs across training iterations. VAEs exhibit frequent and large spikes in both quantities. Importantly, these spikes occur across a wide range of KL regularization weights, suggesting that the instability cannot be eliminated simply by tuning the KL weight. In contrast, G-LZNs remain stable throughout training, with both the training loss and gradient norms evolving smoothly without large spikes. This is consistent with the results in § 4.3, where G-LZNs also remain stable when the additional FD loss is included.

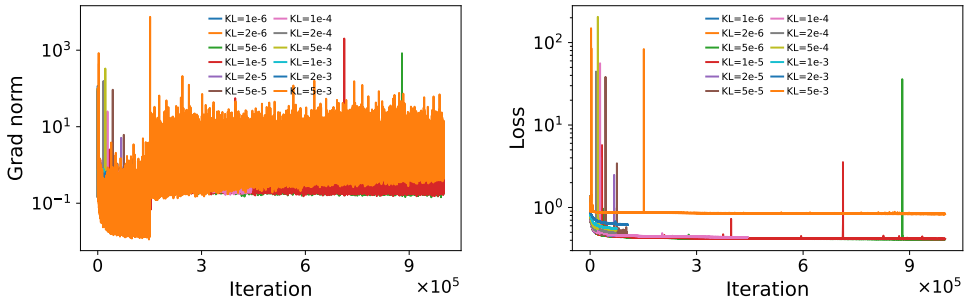


Figure 21: Training loss and gradient norms of VAEs across training iterations on CIFAR10 for different KL regularization weights. VAEs exhibit frequent large spikes in both loss and gradient norm across a wide range of KL weights. The $y$-axis is shown on a logarithmic scale.

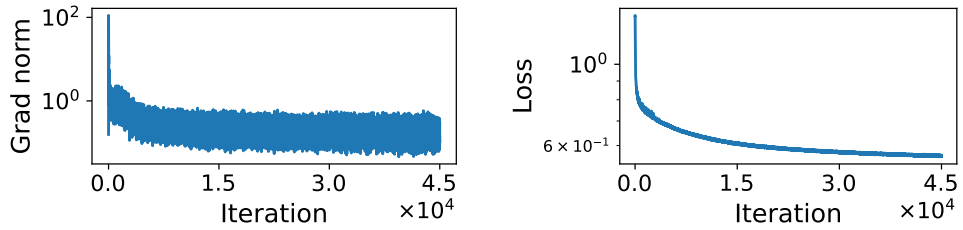


Figure 22: Training loss and gradient norms of G-LZNs across training iterations on CIFAR10. Both quantities evolve smoothly throughout training without large spikes. The $y$-axis is shown on a logarithmic scale.

**G-LZNs have better sample quality than VAEs.** For VAEs, we select the KL regularization weight that achieves the best FID ($\mathrm{KL} = 10^{-3}$) and compare its image-quality metrics with G-LZNs in Tab. 5. G-LZNs perform better on all metrics except precision, where the two methods are comparable. We additionally compute the average squared $\ell_2$ distance between each generated image and its nearest training image as an indicator of memorization. G-LZNs generate samples that are farther from their nearest training examples, suggesting better generalization.

Fig. 23 further shows the VAE FID scores across all KL regularization weights. VAE performance is highly sensitive to the KL weight: values either below or above the optimal $\mathrm{KL} = 10^{-3}$ can substantially degrade sample quality.

Table 5: Generation quality comparison between VAEs and G-LZNs on CIFAR10. FD loss (§ C.1.2) is not used for both methods in this experiment. The main difference between the two models is the use of latent zoning in G-LZNs versus KL regularization in VAEs. G-LZNs performs better on most sample quality metrics while also generating more novel samples, as indicated by the larger distance to the nearest training sample.

| Method | FID ↓ | sFID ↓ | Recall ↑ | Precision ↑ | Inception Score ↑ | Distance to Nearest Training Sample ↑ |
|---|---|---|---|---|---|---|
| VAEs (KL $= 10^{-3}$) | 9.23 | 32.02 | 0.427 | **0.717** | 9.60 | 16.66 |
| G-LZNs | **6.80** | **25.60** | **0.493** | 0.697 | **10.03** | **18.94** |

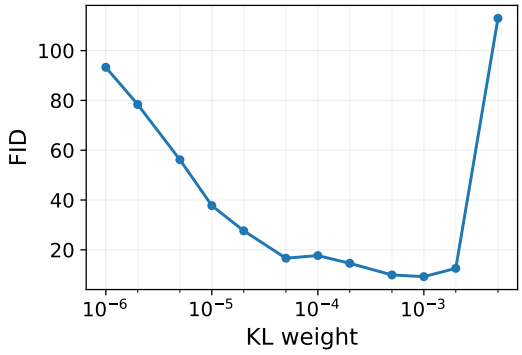


Figure 23: FID across different KL regularization weights for VAEs on CIFAR10. VAE performance is highly sensitive to the KL weight: even modest deviations from the optimal value can substantially degrade sample quality.

## F.9 Ablation Studies and Hyperparameter Analysis

### F.9.1 Sample-wise Reconstruction Loss

Tab. 6 presents the loss ablation results on CIFAR10, highlighting the importance of the sample-level reconstruction loss (§ C.1.1) enabled by our proposed latent-zoning mechanism (§ 3.2).

**Ablating the sample-level reconstruction loss.** The first group of Tab. 6 compares different choices of sample-level reconstruction loss. The main observations are:

- **Simple $\ell_1$ or $\ell_2$ losses work even without pretrained embeddings.** Using only pixel-space $\ell_1$ or $\ell_2$ reconstruction already yields meaningful generation quality. In particular, the $\ell_1$ model achieves an FID of 44.87, already outperforming some previous generative models (Behrmann et al., 2019; Van Den Oord et al., 2016; Ostrovski et al., 2018; Chen et al., 2020; Kingma & Dhariwal, 2018) (see Table 1 of Li et al. (2023)). Fig. 24 shows that the generated samples capture recognizable CIFAR10 objects, indicating that the model learns the data distribution to a substantial degree. The main limitation is blurriness, which is a well-known issue with pixel-wise $\ell_1$ reconstruction.
- **LPIPS substantially improves sharpness and perceptual quality.** LPIPS is widely used in generative modeling to improve perceptual quality and image sharpness Zhang et al. (2018); Esser et al. (2021). Replacing the pixel-space $\ell_1$ or $\ell_2$ loss with LPIPS leads to large improvements across the image-quality metrics, and the generated samples in Fig. 25 are visibly sharper.
- **More modern embedding models further improve image quality.** LPIPS is based on the older VGG architecture. Motivated by recent generative models that use stronger pretrained feature extractors (Deng et al., 2026; Yu et al., 2025), we introduce the improved perceptual loss in § C.1.1, using ConvNeXtV2 (Woo et al., 2023) and DINO (Caron et al., 2021). Both embeddings further improve most image-quality metrics, consistent with the qualitative results in Fig. 26. Tab. 6 also shows that combining ConvNeXtV2 and DINO provides little additional benefit over using either embedding alone. Nevertheless, because our early experiments fixed

the ConvNeXtV2+DINO configuration, we retain this setting for the remaining experiments on both `CIFAR10` and `ImageNet256×256`.

**The group-level reconstruction loss provides additional gains on top of the sample-level loss.** On top of the ConvNeXtV2+DINO sample-level loss, we further add the group-level reconstruction loss (§ C.1.2) based on Inception embeddings. The results are shown in the second group of Tab. 6. Adding the group-level loss further improves most image-quality metrics, and the generated samples in Fig. 5 are also clearly better visually.

**Using only the group-level reconstruction loss does not work.** The third group of Tab. 6 shows that training with only the group-level reconstruction loss (i.e., the Fréchet loss) yields substantially worse image-quality metrics than the full `G-LZNs` model. Fig. 27 further shows that the generated images contain severe artifacts, repeated patterns, and unnatural colors. We make two observations:

- Combined with the previous results, where the sample-level reconstruction loss alone already performs well, these results suggest that the success of `G-LZNs` primarily comes from the sample-level reconstruction objective enabled by our latent-zoning mechanism, rather than from the group-level loss. This highlights the importance of our proposed latent zoning to the overall performance of `G-LZNs`.
- We believe that training with only the Fréchet loss fails because a Fréchet loss of zero does not guarantee that the generated and real distributions are identical: the loss only matches the mean and covariance of their feature distributions. Indeed, prior work introducing this loss for generative modeling (Yang et al., 2026a) only demonstrates its effectiveness for fine-tuning pretrained generative models, rather than for training a generative model from scratch using the Fréchet loss alone.

Table 6: Loss ablation on `CIFAR10`. The row of ConvNeXtV2+DINO without the group-level loss is repeated twice for clarity.

| Reconstruction Loss | | Metrics | | | | |
|---|---|---|---|---|---|---|
| Sample-level Recon. Loss | Group-level Recon. Loss | FID ↓ | sFID ↓ | Recall ↑ | Precision ↑ | Inception Score ↑ |
| *Ablating sample-level reconstruction loss (no group-level loss)* | | | | | | |
| $\ell_2$ | No | 53.65 | 19.68 | 0.390 | 0.533 | 6.42 |
| $\ell_1$ | No | 44.87 | 15.76 | 0.469 | 0.581 | 6.76 |
| LPIPS | No | 10.39 | **9.47** | 0.505 | 0.591 | 8.91 |
| ConvNextV2 | No | 6.96 | 15.79 | 0.502 | **0.720** | 9.88 |
| DINO | No | **5.47** | 10.21 | **0.537** | 0.681 | 9.75 |
| ConvNeXtV2+DINO | No | 6.80 | 25.60 | 0.493 | 0.697 | **10.03** |
| *Ablating group-level reconstruction loss (sample-level loss fixed)* | | | | | | |
| ConvNeXtV2+DINO | No | 6.80 | 25.60 | 0.493 | **0.697** | 10.03 |
| ConvNeXtV2+DINO | Inception | **1.88** | **7.53** | **0.602** | 0.671 | **10.33** |
| *Using only group-level reconstruction loss* | | | | | | |
| No | Inception | 33.18 | 39.45 | 0.090 | 0.490 | 8.11 |

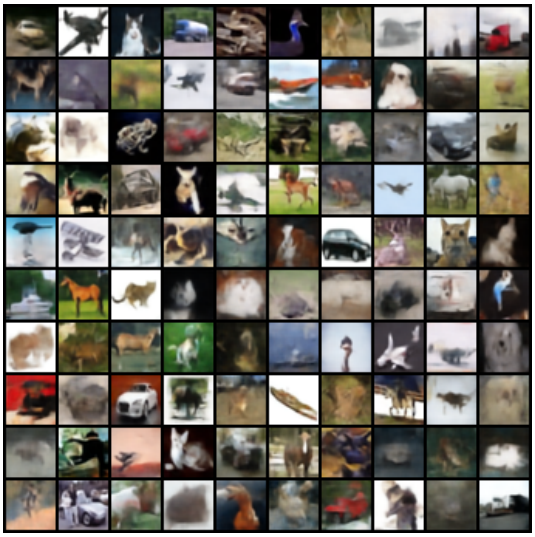

Figure 24: Generated images on `CIFAR10` using only the $\ell_1$ reconstruction loss and no pretrained embeddings (FID = 44.87). The samples capture recognizable objects from `CIFAR10` but tend to appear blurry.

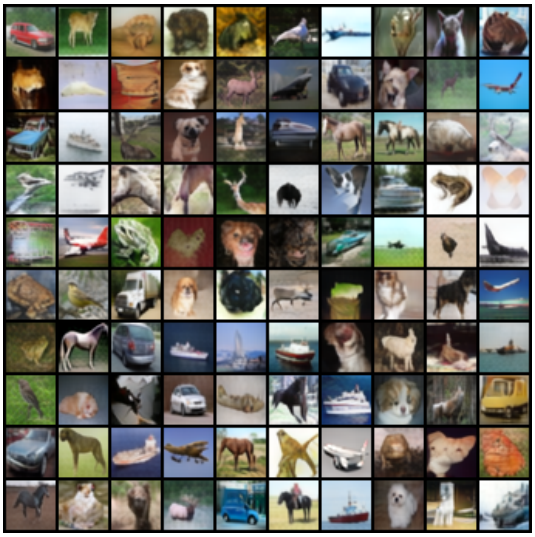

Figure 25: Generated images using only the LPIPS reconstruction loss (FID = 10.39). The samples are sharper and of higher quality than those obtained with the $\ell_1$ loss in Fig. 24.

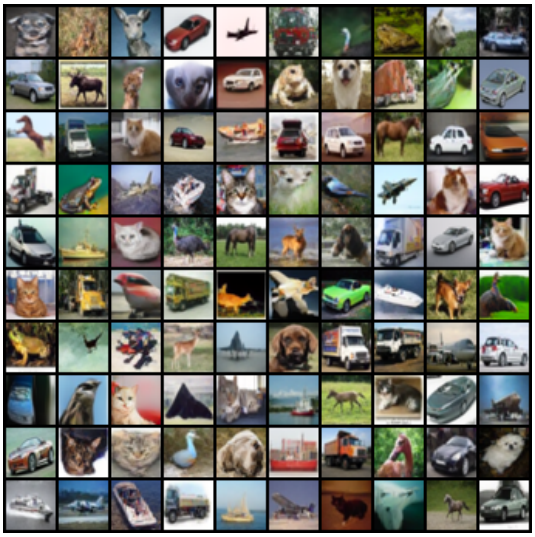

Figure 26: Generated images using ConvNeXtV2+DINO reconstruction loss (FID = 6.80). The samples are sharper and of higher quality than those obtained with the $\ell_1$ loss or LPIPS in Figs. 24 and 25.



Figure 27: Generated images using only the Fréchet loss (FID = 33.18). The samples exhibit severe artifacts, repeated patterns, and unnatural colors. This highlights the importance of our sample-level reconstruction loss (§ C.1.1), enabled by our latent-zoning mechanism (§ 3.2), to the strong performance of G-LZNs.

### F.9.2 GROUP-WISE RECONSTRUCTION LOSS

The ablation study in § F.9.1 shows that the group-wise reconstruction loss improves performance on `CIFAR10`. However, because we use an unconditional generation setting on `CIFAR10`, the group-wise reconstruction loss reduces to the original Fréchet loss of Yang et al. (2026a). In this section, we further ablate the proposed group-wise reconstruction loss (§ C.1.2) on conditional `ImageNet256×256`, where its grouping mechanism can be evaluated directly. We also study how the choice of embedding models used for the group-wise reconstruction loss affects performance.

Tab. 7 shows the ablation results for `G-LZNs` trained with a pretrained autoencoder. Key observations are:

- **Group-wise loss helps.** Comparing Rows 1 and 3 or Rows 2 and 4, applying the group-wise loss consistently improves $\mathrm{FDr}^6$, recall, and precision. Notably, the group-wise loss is less directly aligned with the global $\mathrm{FDr}^6$ metric than the original Fréchet loss, yet it still improves $\mathrm{FDr}^6$. This suggests that the gains come from better matching the underlying conditional distributions rather than simply optimizing toward the evaluation metric.
- **More embedding models help.** Comparing Rows 1 and 2 or Rows 3 and 4, using more embedding models substantially improves $\mathrm{FDr}^6$ while maintaining, and sometimes improving, the other metrics. We therefore use all embedding models in our main experiments.

Table 7: Loss ablation on ImageNet256×256. Group-wise reconstruction loss (§ C.1.2) and additional embedding models both improve image quality. "All" means "SigLIP+Inception+MAE+ConvNeXtV2+DINOv2+CLIP".

| Loss Configuration | | Metrics | | | | | |
|---|---|---|---|---|---|---|---|
| Group-wise | Embedding Models | FDr$^6$ ↓ | FID ↓ | sFID ↓ | Recall ↑ | Precision ↑ | Inception Score ↑ |
| × | SigLIP+Inception+MAE | 6.52 | 1.64 | 57.05 | 0.567 | 0.799 | **341.13** |
| × | All | 3.62 | **1.20** | 60.35 | 0.524 | 0.792 | 338.69 |
| ✓ | SigLIP+Inception+MAE | 5.58 | 1.41 | 67.81 | **0.578** | 0.818 | 327.40 |
| ✓ | All | **2.53** | 1.50 | **55.23** | 0.560 | **0.820** | 333.37 |

### F.9.3 Model Size

We vary model size by changing the base number of feature maps and conduct this study both with and without a pretrained autoencoder. Training without an autoencoder incurs substantially higher memory and computational costs because the high-resolution layers are part of G-LZNs and their parameters must be trained and updated directly. We are therefore only able to scale the number of feature maps to 256 in this setting.

Tab. 8 shows the results. Key observations are:

- As model size increases, most metrics improve, while precision remains similar. This suggests that G-LZNs can benefit from increased model capacity.
- Even with only 45.8M decoder parameters, G-LZNs achieve strong performance, with FDr$^6$ 3.51, outperforming many baselines in Tab. 10. This suggests that G-LZNs make effective use of model capacity, achieving competitive generation quality without relying on heavily overparameterized decoders.

### F.9.4 Whether to Use a Pretrained Autoencoder

We evaluate G-LZNs both with and without a pretrained autoencoder across different model sizes.

Tab. 8 shows that, with the same number of feature maps (256), training without an autoencoder performs better on most metrics, including FDr$^6$, FID, recall, and inception score, while performing worse on sFID and similarly on precision.

This difference may partly result from the larger model size in the no-autoencoder setting. When we increase the number of feature maps in the autoencoder setting to 384, making the decoder parameter counts more comparable (230.0M vs. 182.1M), the two settings each perform better on roughly half of the metrics.

Overall, these results suggest that, at comparable model sizes, G-LZNs achieve similar generation quality with and without a pretrained autoencoder.

Table 8: Model size and autoencoder (AE) ablation on ImageNet256×256. AE indicates whether a pretrained autoencoder is used. Model size refers to the number of parameters in the G-LZN decoder.

| Model Configuration | | | Metrics | | | | | |
|---|---|---|---|---|---|---|---|---|
| AE | #Feature Map | #Param | FDr$^6$ ↓ | FID ↓ | sFID ↓ | Recall ↑ | Precision ↑ | Inception Score ↑ |
| ✓ | 256 | 102.6M | 2.80 | 1.82 | 59.01 | 0.523 | **0.825** | 330.60 |
| ✓ | 384 | 230.0M | 2.53 | **1.50** | **55.23** | 0.560 | 0.820 | 333.37 |
| × | 128 | 45.8M | 3.51 | 2.13 | 81.57 | 0.512 | 0.822 | 319.58 |
| × | 256 | 182.1M | **2.49** | 1.74 | 70.57 | **0.569** | 0.813 | **339.46** |

### F.9.5 Zoning Layers at a Fixed Latent Zone Size

To achieve a target latent zone size $S = {w_1}^l$, we can vary the number of zoning layers $l$ while adjusting the first anchor weight $w_1$ accordingly. Here, we study how the choice of $l$ affects performance while keeping the latent zone size fixed. We set $S = \left(\frac{1}{61}\right)^{15}$ and vary $l$ from 3 to 80. Results are shown in Fig. 28. Key observations are:

- FID remains below 2.7 across the entire range of $l$, demonstrating that G-LZNs are robust to this hyperparameter.
- FID is higher when $l \leq 5$. When $l \geq 10$, FID remains below 2.0 and shows a slight tendency to improve as $l$ increases. We hypothesize that deeper latent zoning allows more complex latent-zone structures, which may improve performance. However, larger $l$ also increases training cost due to the additional latent-zoning computation.

Based on these results, we use $l = 15$ in our main experiments.

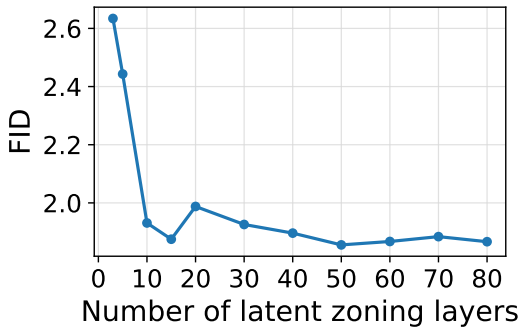


Figure 28: FID on CIFAR10 for different numbers of latent zoning layers $l$, with the latent zone size fixed at $S = \left(\frac{1}{61}\right)^{15}$. Very small $l$ slightly degrades FID, while FID remains consistently low once $l \geq 10$.

### F.9.6 Number of Anchors at a Fixed Latent Zone Size

Building on § F.9.5, we further study how the number of anchors $k$ affects the performance of G-LZNs while keeping the latent zone size $S$, the number of zoning layers $l$, and the first anchor weight $w_1$ fixed. Specifically, we set $w_1 = \frac{1}{61}$ and $l = 15$, which fixes the latent zone size at $S = \left(\frac{1}{61}\right)^{15}$. We vary $k$ from 2 to 13 and assign equal weights to the remaining anchors, each with weight $\frac{1}{k-1} \cdot \frac{60}{61}$. The results are shown in Fig. 29. Key observations are:

- FID remains consistently low ($\leq 2.3$) across the entire range of $k$, demonstrating that G-LZNs are robust to the number of anchors per layer.
- FID also shows a slight tendency to decrease as $k$ increases. We hypothesize that this arises for the same reason as increasing the number of zoning layers at a fixed latent zone size (§ F.9.5): more anchors allow more complex latent-zone structures, which may improve performance.

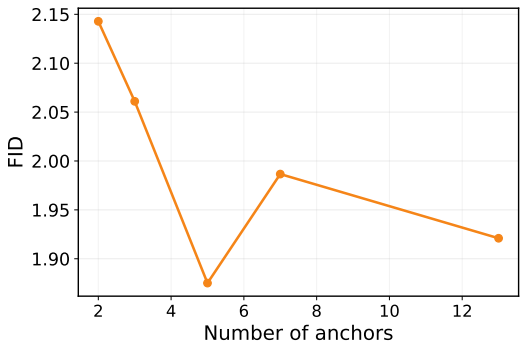


Figure 29: FID on CIFAR10 for different numbers of anchors $k$ per layer, with the latent zone size fixed at $S = \left(\frac{1}{61}\right)^{15}$ and the number of zoning layers fixed at $l = 15$. FID remains consistently low ($\leq 2.3$) as $k$ varies from 2 to 13, demonstrating robustness to the number of anchors per layer. Larger $k$ also shows a slight tendency to improve FID.

### F.9.7 Latent Dimension

In this section, we study the sensitivity of G-LZNs to the latent dimensionality. We vary the latent dimension from 20 to 400 while keeping all other hyperparameters and architectures fixed. Fig. 30 shows that FID remains low ($\leq 2.3$) across this wide range, demonstrating that G-LZNs are robust to the choice of latent dimension. This behavior is expected: similar to VAEs, the latent dimension in G-LZNs primarily sets an upper bound on the dimensionality of the generative distribution, so choosing a latent space larger than necessary does not substantially degrade performance.

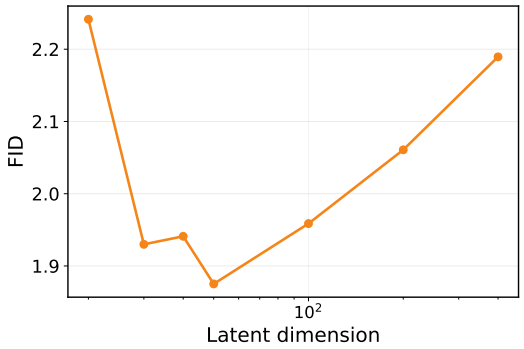


Figure 30: FID on CIFAR10 under different latent dimensions. G-LZNs maintain strong performance (FID $\leq 2.3$) across a wide range of latent dimensions.

### F.10 Extended Image Generation Benchmarks

The main-text tables (Tabs. 2a and 2b) restrict comparisons to methods trained without pretrained generative checkpoints. Here, we broaden the comparison: Tab. 9 adds methods that use or initialize from pretrained generative checkpoints on CIFAR10, while Tab. 10 includes both categories on ImageNet256×256. These extended comparisons show that G-LZNs not only achieve state-of-the-art performance among one-step methods trained without pretrained generative checkpoints, but also outperform many methods that rely on them.

Tabs. 11 and 12 provide the sources for all reported numbers.

Table 9: Unconditional image generation on `CIFAR10` for methods that use or initialize from pretrained generative checkpoints, and are therefore excluded from Tab. 2a. NFE is the number of network evaluations. **Bold** marks the best value among the one-step (NFE 1) models; underline marks the best at any NFE. Sources for all numbers are in Tab. 11.

| | Method | NFE ↓ | Param | FID ↓ | IS ↑ |
|---|---|---|---|---|---|
| GANs | EDM-G++ (2023) | 53 | 107M | 1.77 | — |
| | SiD$^2$A (2024) | 1 | 56M | **1.50** | — |
| | EDM + DDO (2025b) | 35 | 56M | 1.38 | — |
| CMs | ECT (2024) | 1 | 56M | 3.60 | — |
| | | 2 | 56M | 2.11 | — |
| | CTM (2024) | 1 | 59M | 1.98 | — |
| | FACM (2025) | 1 | 56M | 2.69 | — |
| | | 2 | 56M | 1.87 | — |

Table 10: Class-conditional image generation on `ImageNet256×256`, full comparison. The upper block uses or initializes from a pretrained generative checkpoint, by distilling from it or by post-training it, and is therefore not directly comparable to the lower block, which does not. A pretrained *feature* encoder and the VAE/tokenizer that defines the latent space do not count: they are shared infrastructure for this benchmark. #Params counts the generator plus any pretrained decoder or tokenizer. NFE is the number of network evaluations. **Bold** marks the best value among the one-step (NFE 1) models and underline the best at any NFE, both taken within a block, since the two blocks are not comparable. Sources for all numbers are in Tab. 12.

| | Method | Space | NFE | #Params | FD$^6_r$ ↓ | FID↓ | IS↑ | Prec↑ | Rec↑ |
|---|---|---|---|---|---|---|---|---|---|
| *Uses or initializes from pretrained generative checkpoints* | | | | | | | | | |
| DMs / FM | $\pi$-Flow-XL/2 (2026) | lat | 1 | 675M+49M | — | 2.85 | — | — | — |
| | FreeFlow-XL/2 (2025) | lat | 1 | 678M+49M | — | 1.45 | — | — | — |
| AR | DD1, VAR-d24 (2025b) | disc | 1 | 1.09B+65M | — | 8.92 | 202.8 | 0.78 | 0.39 |
| | DD1, VAR-d24 (2025b) | disc | 2 | 1.09B+65M | — | 6.95 | 222.5 | 0.83 | 0.43 |
| | Di[M]O (2025) | disc | 1 | 172M+31M | — | 6.91 | 214.0 | 0.83 | 0.38 |
| | FPD (2026) | disc | 1 | 172M+31M | — | 6.90 | 215.0 | 0.81 | 0.35 |
| | DD2, VAR-d24* (2025a) | disc | 1 | 1.04B+65M | — | 4.91 | 282.2 | **0.87** | 0.39 |
| Others | pMF-L + FD-loss (2026a) | pix | 1 | 410M | 2.09 | 0.78 | 309.2 | 0.76 | 0.67 |
| | JiT-L + FD-loss (2026a) | pix | 1 | 459M | 3.24 | 0.77 | **317.3** | 0.77 | 0.66 |
| | pMF-H + FD-loss (2026a) | pix | 1 | 935M | **1.89** | 0.77 | 310.1 | 0.77 | **0.68** |
| | iMF-XL/2 + FD-loss (2026a) | lat | 1 | 610M+49M | 2.45 | 0.76 | 301.3 | 0.77 | 0.67 |
| | JiT-H + FD-loss (2026a) | pix | 1 | 953M | 2.65 | **0.75** | 313.0 | 0.76 | 0.66 |
| *Does not use or initialize from pretrained generative checkpoints* | | | | | | | | | |
| GANs | BigGAN (2019) | pix | 1 | 112M | — | 6.95 | 152.8 | — | — |
| | GigaGAN (2023) | pix | 1 | 569M | — | 3.45 | 225.5 | — | — |
| | StyleGAN-XL (2022) | pix | 1 | 166M | — | 2.30 | 265.1 | — | — |
| DMs / FM | ADM-G (2021) | pix | 250×2 | 554M | — | 4.59 | 186.7 | — | — |
| | LDM-4-G (2022) | lat | 250 | 400M+33M | — | 3.60 | 247.7 | — | — |
| | JiT-L (2026) | pix | 50×2×2 | 459M | 10.73 | 2.59 | 288.5 | 0.79 | 0.59 |
| | Simple diffusion, UViT/2 (2023) | pix | 1000×2 | 2.5B | — | 2.44 | 256.3 | — | — |
| | DiT-XL/2 (2023) | lat | 250×2 | 675M+49M | — | 2.27 | 278.2 | 0.83 | 0.57 |
| | VDM++, UViT/2 (2023) | pix | 256×2 | 2.5B | — | 2.12 | 267.7 | — | — |
| | SiT-XL/2 (2024) | lat | 250×2 | 675M+49M | 8.44 | 2.12 | 256.7 | 0.81 | 0.60 |
| | PixNerd-XL (2025a) | pix | 100×2 | 1.0B | 5.01 | 2.10 | 318.8 | 0.81 | 0.59 |
| | JiT-H (2026) | pix | 50×2×2 | 953M | 7.66 | 1.97 | 296.0 | 0.78 | 0.63 |
| | JiT-G/16 (2026) | pix | 100×2 | 2B | — | 1.82 | 292.6 | — | — |
| | SiD2, UViT/2 (2025) | pix | 512×2 | — | — | 1.73 | — | — | — |
| | PixelDiT/16 (2026b) | pix | 200×2 | 797M | — | 1.61 | 292.7 | — | — |
| | REG (2025) | lat | 250×2 | 685M+49M | 4.64 | 1.54 | 302.9 | 0.78 | 0.62 |
| | SiT-XL/2 + REPA (2025) | lat | 250×2 | 675M+49M | 5.45 | 1.42 | 306.1 | 0.80 | 0.65 |

| | Method | Space | NFE | #Params | $FD_r^6$ ↓ | FID↓ | IS↑ | Prec↑ | Rec↑ |
|---|---|---|---|---|---|---|---|---|---|
| | LightningDiT-XL/2 (2025) | lat | 250×2 | 675M+70M | 4.57 | 1.42 | 294.3 | 0.80 | 0.64 |
| | SiD2, UViT/1 (2025) | pix | 512×2 | — | — | 1.38 | — | — | — |
| | DDT-XL (2025b) | lat | 250×2 | 675M+49M | 5.70 | 1.26 | 309.3 | 0.79 | 0.66 |
| | REPA-E (2025) | lat | 250×2 | 676M+49M | 3.04 | 1.17 | 298.3 | 0.79 | 0.66 |
| | RAE + $DiT_{DH}$-XL/2 (2025a) | lat | 50×2 | 839M+415M | 3.26 | 1.16 | 261.0 | 0.77 | 0.67 |
| AR | MaskGIT (2022) | disc | 8 | 172M+31M | — | 6.18 | 182.1 | 0.80 | 0.51 |
| AR | VAR-d16 (2024) | disc | 10×2 | 310M+65M | — | 3.30 | 274.4 | 0.84 | 0.51 |
| AR | VAR-d20 (2024) | disc | 10×2 | 600M+65M | — | 2.57 | 302.6 | 0.83 | 0.56 |
| AR | VAR-d24 (2024) | disc | 10×2 | 1.0B+65M | — | 2.09 | 312.9 | 0.82 | 0.59 |
| AR | VAR-d30 (2024) | disc | 10×2 | 2B+65M | 6.70 | 1.97 | 304.6 | 0.82 | 0.59 |
| AR | MAR-L (2024) | lat | 256×2×100 | 478M+38M | 6.68 | 1.80 | 293.4 | 0.80 | 0.60 |
| AR | FlowAR-H (2024) | lat | 50×2 | 1.9B+38M | 6.13 | 1.68 | 274.1 | 0.80 | 0.62 |
| AR | MAR-H (2024) | lat | 256×2×100 | 942M+38M | 5.61 | 1.56 | 299.5 | 0.80 | 0.62 |
| AR | MAR-L + DeTok (2026b) | lat | 256×2×100 | 478M+86M | 5.49 | 1.39 | 306.2 | 0.81 | 0.62 |
| AR | BAR-L (2026a) | disc | 256×2×4 | 1.1B+303M | 3.57 | 1.01 | 281.9 | 0.77 | 0.68 |
| CMs | iCT-XL/2 (2023) | lat | 1 | 675M+49M | — | 34.24 | — | — | — |
| CMs | Shortcut-XL/2 (2025) | lat | 1 | 675M+49M | — | 10.60 | — | — | — |
| CMs | EPG-L/16 (2026) | pix | 1 | 540M | — | 8.82 | — | — | — |
| MF | MeanFlow-XL/2 (2025) | lat | 1 | 676M+49M | — | 3.43 | — | — | — |
| MF | pMF-L (2026) | pix | 1 | 410M | 9.09 | 2.72 | 261.7 | **0.81** | 0.56 |
| MF | pMF-H (2026) | pix | 1 | 935M | 6.87 | 2.29 | 267.2 | 0.80 | 0.59 |
| MF | iMF-XL/2 (2026) | lat | 1 | 610M+49M | 8.39 | 1.82 | 278.9 | 0.78 | 0.63 |
| MF | iMF-XL/2 (2026) | lat | 2 | 610M+49M | 7.48 | 1.61 | 289.1 | 0.79 | 0.63 |
| Others | TiM-XL/2 (2025c) | lat | 1 | 664M+49M | — | 3.26 | 210.3 | — | — |
| Others | $\alpha$-Flow-XL/2+ (2025) | lat | 1 | 676M+49M | — | 2.58 | — | — | — |
| Others | AdvFlow-XL/2 (2026a) | lat | 1 | 673M+49M | — | 2.38 | 284.2 | — | — |
| Others | Drifting, B/16 (2026) | pix | 1 | 134M | — | 1.76 | 299.7 | — | — |
| Others | Drifting, B/2 (2026) | lat | 1 | 133M+49M | — | 1.75 | 263.2 | — | — |
| Others | Drifting, L/2 (2026) | lat | 1 | 463M+49M | 10.92 | 1.53 | 257.2 | 0.79 | 0.63 |
| Others | W-Flow, B/2 (2026) | lat | 1 | 133M+49M | 11.62 | 1.52 | 271.8 | 0.80 | 0.62 |
| Others | Drifting, L/16 (2026) | pix | 1 | 465M | 10.51 | 1.43 | 305.8 | **0.81** | 0.60 |
| Others | W-Flow, L/2 (2026) | lat | 1 | 463M+49M | 10.55 | 1.35 | 272.5 | 0.79 | 0.63 |
| Others | W-Flow, XL/2 (2026) | lat | 1 | 679M+49M | 9.51 | 1.29 | 265.4 | 0.76 | **0.67** |
| | **G-LZNs (ours)** | lat | 1 | 230M+49M | 3.62 | **1.20** | 338.7 | 0.79 | 0.52 |
| | **G-LZNs (ours)** | pix | 1 | 182M | **2.49** | 1.74 | **339.5** | **0.81** | 0.57 |

Table 11: Source of every number in Tab. 2a and Tab. 9. A source cell names where the value was read from, which is often not the paper that introduced the method. Param sources are numbered: *Paper* entries give the location in the cited paper, *Code* entries give the repository and the command we built the model with, and the remaining entries record cross-checks and caveats.

| | Method | NFE | NFE/FID/IS Source | Param Source |
|---|---|---|---|---|
| *Does not use or initialize from pretrained generative checkpoints* | | | | |
| GANs | StyleGAN2 (2020b) | 1 | Tab.2 (Song & Dhariwal, 2023) | (1) Code: `NVlabs/stylegan2-ada-pytorch`, `train.py --cfg=cifar` — built with `c_dim=0`, i.e. unconditional<br>(2) Cross-checked: Tab.1 of (Zheng et al., 2025b) reports StyleGAN2-ADA at 20M (same generator) |
| GANs | StyleGAN2-ADA (2020a) | 1 | Tab.3, unconditional group (Song et al., 2020b) | (1) Code: `NVlabs/stylegan2-ada-pytorch`, `train.py --cfg=cifar` — built with `c_dim=0`, i.e. unconditional<br>(2) Cross-checked: Tab.1 of (Zheng et al., 2025b) reports StyleGAN2-ADA at 20M |
| | NVAE (2021) | 1 | Tab.2 (Song & Dhariwal, 2023) | (1) Code: `NVlabs/NVAE`, `train.py --dataset cifar10 --num_channels_enc 128 --num_channels_dec 128 --num_latent_scales 1 --num_groups_per_scale 30 --num_latent_per_group 20 --num_cell_per_cond_enc 2 --num_cell_per_cond_dec 2 --num_preprocess_blocks 1 --num_postprocess_blocks 1 --num_preprocess_cells 2 --num_postprocess_cells 2 --num_nf 1 --use_se --res_dist` — the CIFAR-10 command in the repo README<br>(2) the full autoencoder comes to 130,852,480; we report the 51,827,336 that `AutoEncoder.sample()` evaluates. No published CIFAR-10 parameter count for NVAE exists to compare against |

| | Method | NFE | NFE/FID/IS Source | Param Source |
|---|---|---|---|---|
| VAEs | CoVAE (2025) | 1 | Tab.5 (Silvestri & Ambrogioni, 2025) | (1) Code: `gisilvs/covae`, `network.model_channels=128 network.channel_mult_enc=[2,2,4] network.channel_mult_dec=[2,2,4] network.num_blocks_enc=4 network.num_blocks_dec=4 network.attn_resolutions=[16,8] network.z_channels=16 dataset.out_channels=6` — the CoVAE CIFAR-10 command in the repo README (the non-adversarial variant)<br>(2) Cross-checked: the full autoencoder comes to 112,899,926, matching the "112M parameters configuration" the paper says it trains; the paper adds that the decoder is about the size of a CIFAR-10 consistency model, and we get 61.7M |
| NFs | Glow (2018) | 1 | Tab.2 (Song & Dhariwal, 2023) | (1) Config: `openai/glow`, `train.py --problem cifar10 --image_size 32 --n_level 3 --depth 32 --flow_permutation 2 --flow_coupling 1 --learntop` — L=3, K=32, width=512, affine coupling, invertible 1x1, unconditional<br>(2) Counted with: `y0ast/Glow-PyTorch` — the official Glow code is TensorFlow 1 |
| | Residual Flow (2020) | 1 | Tab.2 (Song & Dhariwal, 2023) | (1) Code: `rtqichen/residual-flows`, `train_img.py --data cifar10 --actnorm True` — the README command; every other hyper-parameter is the upstream default |
| DMs/FM | DDPM (2020) | 1000 | Tab.3, unconditional group (Song et al., 2020b) | (1) Paper: Tab.1, unconditional column (Zheng et al., 2025b) |
| | Rectified Flow (2022) | 127 | Tab.1(a) (Liu et al., 2022)<br>*1-rectified flow on DDPM++ solved with RK45* | (1) Code: `gnobitab/RectifiedFlow`, `ImageGeneration/configs/rectified_flow/cifar10_rf_gaussian_ddpmpp.py` |
| | Score SDE (2020b) | 2000 | Tab.3, unconditional group (Song et al., 2020b)<br>*NCSN++ cont., VE* | (1) Code: `yang-song/score_sde_pytorch`, `configs/ve/cifar10_ncsnpp_continuous.py`<br>(2) Cross-checked: built with the `NCSNpp` implementation that EDM's re-implementation (`NVlabs/edm`, `training/networks.py: SongUNet`) (Karras et al., 2022) agrees with to within 128 params — checked on the consistency-models CIFAR-10 config, not on this row's own `ch_mult=(1,2,2,2)` config |
| | PFGM (2022) | 110 | Tab.2 (Song & Dhariwal, 2023) | (1) Code: `Newbeeer/poisson_flow`, `configs/poisson/cifar10_ddpmpp_deep.py` |
| | Score SDE (deep) (2020b) | 2000 | Tab.3, unconditional group (Song et al., 2020b)<br>*NCSN++ deep cont., VE* | (1) Code: `yang-song/score_sde_pytorch`, `configs/ve/cifar10_ncsnpp_deep_continuous.py`<br>(2) Cross-checked: Tab.1 of (Zheng et al., 2025b) reports NCSN++ at 108M for the same NFE 2000 / FID 2.20 row |
| | LSGM (2021) | 138 | Tab.1, unconditional column (Zheng et al., 2025b)<br>*NFE 138; iCT Tab.2 says 147, and the LSGM paper states no CIFAR-10 NFE of its own* | (1) Paper: Tab.1, unconditional column (Zheng et al., 2025b) |
| | EDM (2022) | 35 | *FID:* Tab.2 config F, CIFAR-10 uncond VP (Karras et al., 2022)<br>*iCT quotes 2.04 from its own re-implementation; FACM quotes 2.01 unattributed*<br>*IS:* Tab.2, the EDM* row (Song & Dhariwal, 2023)<br>** marks iCT's own re-implementation, whose FID is 2.04; EDM's own paper reports no Inception score* | (1) Code: `NVlabs/edm`, `train.py --cond=0 --arch=ddpmpp --precond=edm` — `--augment` defaults to 0.12, which adds the `augment_dim` input to the network<br>(2) Cross-checked: exact match to the released checkpoint `edm-cifar10-32x32-uncond-vp.pkl` (55,733,891); Tab.1 of (Zheng et al., 2025b) reports 56M |
| | PFGM++ (2023) | 35 | Tab.1, unconditional, D=2048 (Xu et al., 2023)<br>*min FID; the paper also gives 1.93 as the top-3 average* | (1) Code: `Newbeeer/pfgmpp`, `--cond=0 --arch=ncsnpp --pfgmpp=1 --aug_dim=2048` — the options the README's checkpoint table gives for the FID 1.91 model, cifar10-ncsnpp-D-2048<br>(2) Cross-checked: identical to EDM (55,733,891), as expected: `--aug_dim` is PFGM++'s D and does not touch the network |
| CMs | iCT (2023) | 1 | Tab.2 (Song & Dhariwal, 2023) | (1) Config: `openai/consistency_models_cifar10`, `configs/cifar10_ve_ct_adaptive.py`<br>(2) Counted with: `yang-song/score_sde_pytorch` (`NCSNpp`) — the official iCT CIFAR-10 code is JAX<br>(3) Cross-checked: the same config built with EDM's own re-implementation of NCSN++ (`NVlabs/edm`, `training/networks.py: SongUNet`, `--arch=ncsnpp`) (Karras et al., 2022) gives 56,395,523, i.e. 128 params apart |
| | iCT (2023) | 2 | Tab.2 (Song & Dhariwal, 2023) | (1) Config: `openai/consistency_models_cifar10`, `configs/cifar10_ve_ct_adaptive.py`<br>(2) Counted with: `yang-song/score_sde_pytorch` (`NCSNpp`) — the official iCT CIFAR-10 code is JAX<br>(3) Cross-checked: the same config built with EDM's own re-implementation of NCSN++ (`NVlabs/edm`, `training/networks.py: SongUNet`, `--arch=ncsnpp`) (Karras et al., 2022) gives 56,395,523, i.e. 128 params apart |
| | iCT-deep (2023) | 1 | Tab.2 (Song & Dhariwal, 2023) | (1) Config: `openai/consistency_models_cifar10`, `configs/cifar10_ve_ct_adaptive.py` — with `num_res_blocks` raised from 4 to 8, the deep variant the iCT paper describes<br>(2) Counted with: `yang-song/score_sde_pytorch` (`NCSNpp`) — the official iCT CIFAR-10 code is JAX |
| | iCT-deep (2023) | 2 | Tab.2 (Song & Dhariwal, 2023) | (1) Config: `openai/consistency_models_cifar10`, `configs/cifar10_ve_ct_adaptive.py` — with `num_res_blocks` raised from 4 to 8, the deep variant the iCT paper describes<br>(2) Counted with: `yang-song/score_sde_pytorch` (`NCSNpp`) — the official iCT CIFAR-10 code is JAX |
| | sCT (2025) | 1 | Tab.1, arXiv v2 (Lu & Song, 2025)<br>*v1 reports 2.97, which IMM and MeanFlow quote* | (1) Inferred: the paper says only "we follow the settings of Score SDE on CIFAR10" and releases neither a parameter count nor a CIFAR-10 configuration, so this reuses our Score SDE NCSN++ cont. count |
| | sCT (2025) | 2 | Tab.1 (Lu & Song, 2025) | (1) Inferred: see sCT NFE 1 |

| | Method | NFE | NFE/FID/IS Source | Param Source |
|---|---|---|---|---|
| Others | IGEBM (2020) | 60 | Tab.2 (Song & Dhariwal, 2023) | (1) Config: `openai/ebm_code_release`, `train.py --exp=cifar10_uncond --dataset=cifar10 --num_steps=60 --large_model` — `--large_model` selects `ResNet32Large` with `num_filters=128`<br>(2) Counted with: a PyTorch transcription of the TensorFlow weight declarations, written for this work — the official IGEBM code is TensorFlow 1 and cannot be run here<br>(3) Cross-checked: hand-summed from the TF weight declarations in `utils.py`, block by block, gives 18,606,727 — identical to the port; and the config's `--num_steps=60` matches the NFE 60 the table quotes |
| | MeanFlow (2025) | 1 | Tab.3 (Geng et al., 2025) | (1) Paper: Tab.3 (Geng et al., 2025) (*the paper states "the same U-net developed from [44] (~55M), applied directly on the pixel space"*) |
| | Sphere Encoder (2026) | 1 | Tab.1, unconditional generation (Yue et al., 2026) | (1) Code: `facebookresearch/sphere-encoder`, `scripts/train_cifar10.sh --image_size 32 --compression_ratio 1.5 --latent_resolution high --pixel_head_type conv --vit_enc_latent_mlp_mixer_depth 2 --vit_dec_latent_mlp_mixer_depth 2 --vit_enc_model_size large --vit_dec_model_size large` — the model size is set to large to match the paper's Tab.16; the released script passes base, which is a smaller model than the paper reports<br>(2) Cross-checked: rebuilding the class-conditional cifar-10 column of the paper's Tab.16 gives 921,569,387, matching its 921M exactly; the unconditional model drops the adaLN class modulation, leaving 615M total of which the decoder is 311M |
| | Sphere Encoder (2026) | 6 | Tab.1, unconditional generation (Yue et al., 2026) | (1) Code: `facebookresearch/sphere-encoder`, `scripts/train_cifar10.sh --image_size 32 --compression_ratio 1.5 --latent_resolution high --pixel_head_type conv --vit_enc_latent_mlp_mixer_depth 2 --vit_dec_latent_mlp_mixer_depth 2 --vit_enc_model_size large --vit_dec_model_size large` — the model size is set to large to match the paper's Tab.16; the released script passes base, which is a smaller model than the paper reports<br>(2) Cross-checked: rebuilding the class-conditional cifar-10 column of the paper's Tab.16 gives 921,569,387, matching its 921M exactly; the unconditional model drops the adaLN class modulation, leaving 615M total of which the decoder is 311M |
| | IMM (2025) | 1 | Tab.1 (Zhou et al., 2025) | (1) Code: `lumalabs/imm`, `configs/cifar10.yaml`<br>(2) Cross-checked: `IMMPrecond.__init__` in `training/preconds.py` prints its own parameter count on construction, and reports `# Mparams: 56.062595` |
| | IMM (2025) | 2 | Tab.1 (Zhou et al., 2025) | (1) Code: `lumalabs/imm`, `configs/cifar10.yaml`<br>(2) Cross-checked: `IMMPrecond.__init__` in `training/preconds.py` prints its own parameter count on construction, and reports `# Mparams: 56.062595` |
| *Uses or initializes from pretrained generative checkpoints* | | | | |
| GANs | EDM-G++ (2023) | 53 | Tab.1, unconditional column (Zheng et al., 2025b) | (1) Paper: Tab.1, unconditional column (denoiser + discriminator, both run at sampling time) (Zheng et al., 2025b) |
| GANs | SiD$^2$A (2024) | 1 | Tab.1, unconditional column (Zheng et al., 2025b) | (1) Paper: Tab.1, unconditional column (Zheng et al., 2025b) |
| GANs | EDM + DDO (2025b) | 35 | Tab.1, unconditional column (Zheng et al., 2025b) | (1) Paper: Tab.1, unconditional column (Zheng et al., 2025b) |
| CMs | ECT (2024) | 1 | Tab.1, the ECM (400k iters) row (Geng et al., 2024)<br>*the table names the model ECM; ECT is the training method* | (1) Code: `locuslab/ect`, `run_ecm.sh --cond=0 --arch=ddpmpp --dropout=0.2 --augment=0.0 --transfer=edm-cifar10-32x32-uncond-vp.pkl`<br>(2) Cross-checked: equals the EDM count minus the 1,152 `augment_dim` params, as expected since `run_ecm.sh` passes `--augment=0.0` |
| CMs | ECT (2024) | 2 | Tab.1, the ECM (400k iters) row (Geng et al., 2024)<br>*the table names the model ECM; ECT is the training method* | (1) Code: `locuslab/ect`, `run_ecm.sh --cond=0 --arch=ddpmpp --dropout=0.2 --augment=0.0 --transfer=edm-cifar10-32x32-uncond-vp.pkl`<br>(2) Cross-checked: equals the EDM count minus the 1,152 `augment_dim` params, as expected since `run_ecm.sh` passes `--augment=0.0` |
| CMs | CTM (2024) | 1 | Tab.1, unconditional column (Zheng et al., 2025b) | (1) Paper: Tab.1, unconditional column (Zheng et al., 2025b) |
| CMs | FACM (2025) | 1 | Tab.1 (Peng et al., 2025) | (1) Inferred: the paper says only "a DDPM++ model in pixel-space", and the official repo ali-vilab/FACM ships no CIFAR-10 configuration, so this reuses the standard DDPM++ CIFAR-10 count |
| CMs | FACM (2025) | 2 | Tab.1 (Peng et al., 2025) | (1) Inferred: see FACM NFE 1 |

Table 12: Source of every number in Tab. 2b and Tab. 10. A source cell names where the value was read from, which is often not the paper that introduced the method. Where the generator and the decoder are documented separately, each is labelled. *Code* entries are counts we made ourselves, and give the repository and the configuration the model was built from.

| Method | NFE | NFE/FD$^6_r$/FID/IS/Prec/Rec Source | Param Source |
|---|---|---|---|
| *Uses or initializes from pretrained generative checkpoints* | | | |
| $\pi$-Flow-XL/2 (2026) | 1 | Tab.2 (Tong et al., 2025)<br>*matches the 1-NFE FID 2.85 in $\pi$-Flow's own abstract* | *generator:* Tab.2 (Tong et al., 2025)<br>*decoder:* Code: `stabilityai/sd-vae-ft-mse`, `AutoencoderKL.decoder` + `post_quant_conv`<br>*SD-VAE ft-mse decoder 49,490,199* |

| | Method | NFE | NFE/FD$^{6}_{r}$/FID/IS/Prec/Rec Source | Param Source |
|---|---|---|---|---|
| DMs / FM | FreeFlow-XL/2 (2025) | 1 | Tab.2 (Tong et al., 2025)<br>*the table reports FID only* | *generator:* Tab.2 (Tong et al., 2025)<br>*decoder:* Code: `stabilityai/sd-vae-ft-mse`, `AutoencoderKL.decoder` + `post_quant_conv`<br>*SD-VAE ft-mse decoder 49,490,199, as for every SiT-XL/2 derivative in this table* |
| AR | DD1, VAR-d24 (2025b) | 1 | Tab.1 (Liu et al., 2025b) | *generator:* Tab.1 (Liu et al., 2025b)<br>*decoder:* Code: `FoundationVision/VAR`, `VQVAE(vocab_size=4096, z_channels=32, ch=160, share_quant_resi=4)`, the released `vae_ch160v4096z32.pth`<br>*unchanged by distillation* |
| | DD1, VAR-d24 (2025b) | 2 | Tab.1 (Liu et al., 2025b) | *generator:* Tab.1 (Liu et al., 2025b)<br>*decoder:* Code: `FoundationVision/VAR`, `VQVAE(vocab_size=4096, z_channels=32, ch=160, share_quant_resi=4)`, the released `vae_ch160v4096z32.pth`<br>*unchanged by distillation* |
| | Di[M]O (2025) | 1 | Tab.1 (Zhu et al., 2025) | *generator:* Code: `google-research/maskgit`, `maskgit_class_cond_config`, transformer built as in `inference.py`<br>*MaskGIT transformer 172,457,193: vocab 2025 (codebook 1024 + 1000 classes + mask token), 24 layers, hidden 768. W-Flow Tab.3 gives 227M, Di[M]O's table gives 174M*<br>*decoder:* Tab.3 (Han et al., 2026) |
| | FPD (2026) | 1 | Tab.2, MaskGIT teacher (Wang & Tong, 2026) | *same as Di[M]O*<br>*decoder:* Tab.3 (Han et al., 2026)<br>*its Tab.2 has no parameter column, but the paper states it initializes "the student from pre-trained MaskGIT", so the generator is MaskGIT's* |
| | DD2, VAR-d24* (2025a) | 1 | Tab.1 (Liu et al., 2025a)<br>*the VAR-d24* row; the asterisk is the paper's own and means "results obtained with more training"* | *generator:* Tab.1 (Liu et al., 2025a)<br>*same as DD1, VAR-d24* |
| Others | pMF-L + FD-loss (2026a) | 1 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | Tab.4, unified evaluation pipeline (Yang et al., 2026a) |
| | JiT-L + FD-loss (2026a) | 1 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | Tab.4, unified evaluation pipeline (Yang et al., 2026a) |
| | pMF-H + FD-loss (2026a) | 1 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | Tab.4, unified evaluation pipeline (Yang et al., 2026a) |
| | iMF-XL/2 + FD-loss (2026a) | 1 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Tab.3 (Han et al., 2026) |
| | JiT-H + FD-loss (2026a) | 1 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | Tab.4, unified evaluation pipeline (Yang et al., 2026a) |
| *Does not use or initialize from pretrained generative checkpoints* | | | | |
| GANs | BigGAN (2019) | 1 | Tab.3 (Han et al., 2026) | Tab.3 (Han et al., 2026) |
| | GigaGAN (2023) | 1 | Tab.3 (Han et al., 2026) | Tab.3 (Han et al., 2026) |
| | StyleGAN-XL (2022) | 1 | Tab.3 (Han et al., 2026) | Tab.3 (Han et al., 2026) |
| | ADM-G (2021) | 250×2 | Tab.3 (Han et al., 2026) | Tab.3 (Han et al., 2026) |
| | LDM-4-G (2022) | 250 | Tab.1 (Tian et al., 2024) | *generator:* Tab.1 (Tian et al., 2024)<br>*decoder:* Code: `CompVis/latent-diffusion`, `configs/latent-diffusion/cin256-v2.yaml`, first stage (vq-f4)<br>*vq-f4 first stage from cin256-v2.yaml: decoder 32,960,771 + post_quant_conv 12 + codebook 24,576 = 32,985,359* |
| | JiT-L (2026) | 50×2×2 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | Tab.4, unified evaluation pipeline (Yang et al., 2026a) |
| | Simple diffusion, UViT/2 (2023) | 1000×2 | Tab.6 (Deng et al., 2026) | Tab.6 (Deng et al., 2026) |
| | DiT-XL/2 (2023) | 250×2 | Tab.1 (Prec/Recall) (Tian et al., 2024)<br>Tab.3 (FID/IS) (Han et al., 2026) | Tab.3 (Han et al., 2026) |
| | VDM++, UViT/2 (2023) | 256×2 | Tab.6 (Deng et al., 2026) | Tab.6 (Deng et al., 2026) |
| | SiT-XL/2 (2024) | 250×2 | Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*W-Flow Tab.3 reports FID 2.06 / IS 270. 3 for this model* | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Tab.3 (Han et al., 2026) |
| | PixNerd-XL (2025a) | 100×2 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | Tab.4, unified evaluation pipeline (Yang et al., 2026a) |

| | Method | NFE | NFE/FD$^{6}_{r}$/FID/IS/Prec/Rec Source | Param Source |
|---|---|---|---|---|
| DMs / FM | JiT-H (2026) | 50×2×2 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | Tab.4, unified evaluation pipeline (Yang et al., 2026a) |
| | JiT-G/16 (2026) | 100×2 | Tab.6 (Deng et al., 2026) | Tab.6 (Deng et al., 2026) |
| | SiD2, UViT/2 (2025) | 512×2 | Tab.6 (Deng et al., 2026) | Tab.6 (Deng et al., 2026) |
| | PixelDiT/16 (2026b) | 200×2 | Tab.6 (Deng et al., 2026) | Tab.6 (Deng et al., 2026) |
| | REG (2025) | 250×2 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Tab.3 (Han et al., 2026) |
| | SiT-XL/2 + REPA (2025) | 250×2 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Tab.3 (Han et al., 2026) |
| | LightningDiT-XL/2 (2025) | 250×2 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Tab.3 (VA-VAE) (Han et al., 2026) |
| | SiD2, UViT/1 (2025) | 512×2 | Tab.6 (Deng et al., 2026) | Tab.6 (Deng et al., 2026) |
| | DDT-XL (2025b) | 250×2 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Tab.3 (Han et al., 2026) |
| | REPA-E (2025) | 250×2 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Tab.3 (Han et al., 2026) |
| | RAE + DiT$_{\mathrm{DH}}$-XL/2 (2025a) | 50×2 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Tab.3 (RAE) (Han et al., 2026) |
| AR | MaskGIT (2022) | 8 | Tab.1 (Prec/Recall) (Tian et al., 2024)<br>Tab.3 (FID/IS) (Han et al., 2026) | *same as Di[M]O*<br>Tab.3 (Han et al., 2026) |
| | VAR-d16 (2024) | 10×2 | Tab.1 (Tian et al., 2024) | *generator:* Tab.1 (Tian et al., 2024)<br>*decoder:* Code: `FoundationVision/VAR`, `VQVAE(vocab_size=4096, z_channels=32, ch=160, share_quant_resi=4)`, the released `vae_ch160v4096z32.pth`<br>*vae_ch160v4096z32: decoder 64,653,923 + post_quant_conv 9,248 + quantize 168,064 = 64,831,235, confirmed against the released checkpoint; the full VQVAE comes to 108.9M, matching VAR's stated 109M. W-Flow Tab.3 gives 61M* |
| | VAR-d20 (2024) | 10×2 | Tab.1 (Tian et al., 2024) | *generator:* Tab.1 (Tian et al., 2024)<br>*same as VAR-d16* |
| | VAR-d24 (2024) | 10×2 | Tab.1 (Tian et al., 2024) | *generator:* Tab.1 (Tian et al., 2024)<br>*same as VAR-d16* |
| | VAR-d30 (2024) | 10×2 | Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*VAR Tab.1 reports FID 1.92 / IS 323.1 for this model* | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*same as VAR-d16* |
| | MAR-L (2024) | 256×2×100 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Code: `LTH14/mar`, `AutoencoderKL(embed_dim=16, ch_mult=(1,1,2,2,4))`, i.e. `kl16.ckpt`<br>*KL-16 VAE: decoder 38,188,931 + post_quant_conv 272, confirmed against the official kl16.safetensors. W-Flow Tab.3 gives 41M* |
| | FlowAR-H (2024) | 50×2 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Code: `LTH14/mar`, `AutoencoderKL(embed_dim=16, ch_mult=(1,1,2,2,4))`, i.e. `kl16.ckpt`<br>*eval.py loads MAR's kl16.ckpt; decoder 38,188,931 + post_quant_conv 272* |
| | MAR-H (2024) | 256×2×100 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Code: `LTH14/mar`, `AutoencoderKL(embed_dim=16, ch_mult=(1,1,2,2,4))`, i.e. `kl16.ckpt`<br>*same KL-16 VAE as MAR-L; W-Flow Tab.3 lists this row as 943M+41M* |

| | Method | NFE | NFE/FD$^{6}_{r}$/FID/IS/Prec/Rec Source | Param Source |
|---|---|---|---|---|
| | MAR-L + DeTok (2026b) | 256×2×100 | Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*The DeTok README gives FID 1.43 for DeTok-BB and 1.32 for DeTok-BB-decoder_ft; FD-loss Tab.4's unified 1.39 does not say which* | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Code: `Jiawei-Yang/DeTok`, `detok_BB(img_size=256, patch_size=16, token_channels=16)`<br>*detok_BB decoder 85,850,964; the full tokenizer comes to 171,712,628, matching the 172M in the repo README* |
| | BAR-L (2026a) | 256×2×4 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Code: `amazon-far/BAR`, `configs/generator/bar_l.yaml`, `SigLIP2Decoder` + FSQ `out_proj`<br>*SigLIP2Decoder (ViT-L) 303,364,948 + FSQ out_proj 17,408 = 303,382,356; FSQ is a fixed quantizer, so there is no codebook to count* |
| CMs | iCT-XL/2 (2023) | 1 | Tab.3 (Han et al., 2026) | Tab.3 (Han et al., 2026) |
| CMs | Shortcut-XL/2 (2025) | 1 | Tab.3 (Han et al., 2026) | Tab.3 (Han et al., 2026) |
| CMs | EPG-L/16 (2026) | 1 | Tab.6 (Deng et al., 2026) | Tab.6 (Deng et al., 2026) |
| MF | MeanFlow-XL/2 (2025) | 1 | Tab.3 (Han et al., 2026) | Tab.3 (Han et al., 2026) |
| MF | pMF-L (2026) | 1 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | Tab.4, unified evaluation pipeline (Yang et al., 2026a) |
| MF | pMF-H (2026) | 1 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | Tab.4, unified evaluation pipeline (Yang et al., 2026a) |
| MF | iMF-XL/2 (2026) | 1 | Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*W-Flow Tab.3 reports FID 1.72 / IS 282. 0 for this model* | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Tab.3 (Han et al., 2026) |
| MF | iMF-XL/2 (2026) | 2 | Tab.4, unified evaluation pipeline (Yang et al., 2026a) | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Tab.3 (Han et al., 2026) |
| Others | TiM-XL/2 (2025c) | 1 | Tab.3 (Han et al., 2026) | Tab.3 (Han et al., 2026) |
| Others | $\alpha$-Flow-XL/2+ (2025) | 1 | Tab.3 (Han et al., 2026) | Tab.3 (Han et al., 2026) |
| Others | AdvFlow-XL/2 (2026a) | 1 | Tab.3 (Han et al., 2026) | Tab.3 (Han et al., 2026) |
| Others | Drifting, B/16 (2026) | 1 | Tab.6 (Deng et al., 2026) | Tab.6 (Deng et al., 2026) |
| Others | Drifting, B/2 (2026) | 1 | Tab.5 (Deng et al., 2026) | Tab.5 (Deng et al., 2026) |
| Others | Drifting, L/2 (2026) | 1 | Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*Drifting Tab.5 reports FID 1.54 / IS 258. 9 for this model* | *generator:* Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*decoder:* Tab.5 (Deng et al., 2026) |
| Others | W-Flow, B/2 (2026) | 1 | Tab.3 (FID/IS) (Han et al., 2026)<br>measured by us: FDr6 over 50,000 samples from the released checkpoint (`jiaqihan99/W-Flow`, `latent_sota_B_ot`, CFG 1.19 as in the paper's own eval script), scored with the six-model protocol of (Yang et al., 2026a)<br>measured by us: Precision/Recall over the same 50,000 samples against the ADM reference batch `VIRTUAL_imagenet256_labeled.npz`<br>*the same run reproduces the paper's FID to within 0.02 (1.54 against 1.52)* | Tab.3 (Han et al., 2026) |
| Others | Drifting, L/16 (2026) | 1 | Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>*Drifting Tab.6 reports FID 1.61 / IS 307. 5 for this model; the value here is the FD-loss paper's re-evaluation* | Tab.4, unified evaluation pipeline (Yang et al., 2026a)<br>Tab.6 gives 464M instead (Deng et al., 2026) |
| Others | W-Flow, L/2 (2026) | 1 | Tab.3 (FID/IS) (Han et al., 2026)<br>measured by us: FDr6 over 50,000 samples from the released checkpoint (`jiaqihan99/W-Flow`, `latent_sota_L_ot`, CFG 1.14 as in the paper's own eval script), scored with the six-model protocol of (Yang et al., 2026a)<br>measured by us: Precision/Recall over the same 50,000 samples against the ADM reference batch `VIRTUAL_imagenet256_labeled.npz`<br>*the same run reproduces the paper's FID to within 0.03 (1.38 against 1.35)* | Tab.3 (Han et al., 2026) |

| Method | NFE | NFE/FD$^{6}_{r}$/FID/IS/Prec/Rec Source | Param Source |
|---|---|---|---|
| W-Flow, XL/2 (2026) | 1 | Tab.3 (FID/IS) (Han et al., 2026)<br>measured by us: FDr6 over 50,000 samples from the released checkpoint (`jiaqihan99/W-Flow`, `latent_sota_XL_ot`, CFG 1.09 as in the paper's own eval script), scored with the six-model protocol of (Yang et al., 2026a)<br>measured by us: Precision/Recall over the same 50,000 samples against the ADM reference batch `VIRTUAL_imagenet256_labeled.npz`<br>*the same run reproduces the paper's FID to within 0.06 (1.35 against 1.29)* | Tab.3 (Han et al., 2026) |