Direct FID as Loss: From Gradient Computation to Streaming Training

Su Jianlin

2026-05-08

Abstract

Readers familiar with visual generation models know that FID has long been one of the key evaluation metrics, with lower values typically indicating more realistic generation. A natural question arises: why not directly use FID as the loss function for training generative models? Is it because FID is non-differentiable? Not at all—FID is in fact differentiable, and in theory presents no issue as a loss function. However, practical difficulties in computation arise. Recently, the paper *Representation Fréchet Loss for Visual Generation* has made attempts to overcome these challenges, successfully applying FID to fine-tune generative models and significantly improving single-step generation performance. This article briefly explores the underlying mathematics and implementation techniques.

1 Generation Metrics

FID, short for "Fréchet Inception Distance", can be understood in two parts: "Fréchet Distance (FD)" and "Inception (I)". Suppose we have two distributions p and q, representing real and generated samples respectively. We encode input samples $\boldsymbol{x}$ via a pretrained encoder ϕ into feature vectors $\boldsymbol{z} = \phi(\boldsymbol{x}) \in \mathbb{R}^d$, and estimate their mean vectors $\boldsymbol{\mu}_p, \boldsymbol{\mu}_q$ and covariance matrices $\boldsymbol{\Sigma}_p, \boldsymbol{\Sigma}_q$. Assuming the encoded features follow multivariate normal distributions, we can measure their discrepancy using the Wasserstein-2 distance:

$$\begin{aligned}\mathcal{F} \triangleq \mathcal{W}_2^2[p,q] &= \|\boldsymbol{\mu}_p - \boldsymbol{\mu}_q\|^2 + \mathrm{tr}(\boldsymbol{\Sigma}_p + \boldsymbol{\Sigma}_q - 2(\boldsymbol{\Sigma}_p\boldsymbol{\Sigma}_q)^{1/2}) \\ &= \|\boldsymbol{\mu}_p - \boldsymbol{\mu}_q\|^2 + \mathrm{tr}(\boldsymbol{\Sigma}_p + \boldsymbol{\Sigma}_q - 2(\boldsymbol{\Sigma}_p^{1/2}\boldsymbol{\Sigma}_q\boldsymbol{\Sigma}_p^{1/2})^{1/2})\end{aligned} \tag{1}$$

where we define tr as the trace of a matrix. Substituting the estimated mean and covariance from the encoded features into this equation gives the "Fréchet Distance (FD)". For details on the derivation of this formula, interested readers may refer to *KL Divergence, Bhattacharyya Distance, and Wasserstein Distance between Two Multivariate Normal Distributions*.

When the encoder ϕ is taken to be InceptionV3 (I), the resulting metric is called "Fréchet Inception Distance", or FID. This evaluation metric was first introduced in the 2017 paper *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*, which in some sense belongs to the "ancient era" of generative modeling.

Nowadays, however, in both training and evaluation, InceptionV3 is not mandatory; more advanced feature extractors such as SigLIP can be used instead. Alternatively, one may compute the Fréchet Distance using multiple distinct encoders and sum the results. We collectively refer to such generalizations as "FD Loss".

2 Related Work

Although FID appears complex, it involves no non-differentiable operations, making its use as a loss function a natural idea. Several years ago, such attempts already existed, for example, in *Image Generation Via Minimizing Fréchet Distance in Discriminator Feature Space* and *Backpropagating through Fréchet Inception Distance*.

However, early attempts did not achieve impressive results. The fundamental reason lies in batch size. Typical loss functions compute per-sample losses and then average them. In contrast, FID computes means and covariances over the entire batch before evaluating the nonlinear function in (1). Consequently, FID estimated with small batches is biased, and this bias cannot be eliminated through prolonged training, only mitigated by increasing the batch size—leading to prohibitively high training costs.

The phrases "nonlinear operations involving cross-sample statistics" and "requirement for large batch size" may sound familiar. Indeed, contrastive learning in vision typically shares these two characteristics. Due to nonlinear interactions across samples, gradient accumulation cannot easily increase effective batch size—but this is not impossible to solve. See for example *Can Gradient Accumulation Be Used in Contrastive Learning?*. As we shall see, the solution to the batch size problem in FID is conceptually similar.

From another perspective—using pretrained models to extract features for loss construction—there is also the related work on *Perceptual Loss*. However, this serves as a reconstruction loss for individual samples and is typically used in training models like VAEs. It does not involve cross-sample statistical computations, and thus presents no computational difficulties.

3 Gradient Computation

We now proceed step by step to derive the gradients and understand the challenges in using FD as a loss. First, we need to compute the gradients. Since p represents the real data distribution, the statistics $\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p$ are fixed, and we only need to compute gradients with respect to $\boldsymbol{\mu}_q, \boldsymbol{\Sigma}_q$.

The gradient with respect to $\boldsymbol{\mu}_q$ is straightforward:

$$\nabla_{\boldsymbol{\mu}_q} \mathcal{F} = \nabla_{\boldsymbol{\mu}_q} \|\boldsymbol{\mu}_p - \boldsymbol{\mu}_q\|^2 = 2(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p) \tag{2}$$

The gradient with respect to $\boldsymbol{\Sigma}_q$ is:

$$\nabla_{\boldsymbol{\Sigma}_q} \mathcal{F} = \nabla_{\boldsymbol{\Sigma}_q} \operatorname{tr}(\boldsymbol{\Sigma}_p + \boldsymbol{\Sigma}_q - 2(\boldsymbol{\Sigma}_p^{1/2}\boldsymbol{\Sigma}_q\boldsymbol{\Sigma}_p^{1/2})^{1/2}) = \boldsymbol{I} - 2\nabla_{\boldsymbol{\Sigma}_q} \operatorname{tr}((\boldsymbol{\Sigma}_p^{1/2}\boldsymbol{\Sigma}_q\boldsymbol{\Sigma}_p^{1/2})^{1/2}) \tag{3}$$

We use the second line of (1), which is more complex but has an advantage: the matrix $\boldsymbol{S} = \boldsymbol{\Sigma}_p^{1/2}\boldsymbol{\Sigma}_q\boldsymbol{\Sigma}_p^{1/2}$ is symmetric positive definite, a property that enables simplification. Let the eigen-decomposition of $\boldsymbol{S}$ be $\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$, so $\boldsymbol{S}^{1/2} = \boldsymbol{U}\boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^\top$. Then:

$$\operatorname{tr}(\boldsymbol{S}^{1/2}) = \operatorname{tr}(\boldsymbol{\Lambda}^{1/2}) = \sqrt{\lambda_1} + \sqrt{\lambda_2} + \cdots + \sqrt{\lambda_d} \tag{4}$$

$$\nabla_{\boldsymbol{S}} \operatorname{tr}(\boldsymbol{S}^{1/2}) = \frac{1}{2}\sum_{i=1}^{d} \frac{\nabla_{\boldsymbol{S}}\lambda_i}{\sqrt{\lambda_i}} = \frac{1}{2}\sum_{i=1}^{d} \frac{\boldsymbol{u}_i\boldsymbol{u}_i^\top}{\sqrt{\lambda_i}} = \frac{1}{2}\boldsymbol{U}\boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^\top = \frac{1}{2}\boldsymbol{S}^{-1/2} \tag{5}$$

For derivations of eigenvalue gradients, refer to *Derivatives of SVD*. The result resembles the derivative of $\sqrt{x}$, namely $\frac{1}{2\sqrt{x}}$, which feels intuitive. However, this is not trivial and generally fails if $\boldsymbol{S}$ is not symmetric positive definite. Finally, by the chain rule:

$$\nabla_{\boldsymbol{\Sigma}_q} \operatorname{tr}(\boldsymbol{S}^{1/2}) = \boldsymbol{\Sigma}_p^{1/2}[\nabla_{\boldsymbol{S}} \operatorname{tr}(\boldsymbol{S}^{1/2})]\boldsymbol{\Sigma}_p^{1/2} = \frac{1}{2}\boldsymbol{\Sigma}_p^{1/2}\boldsymbol{S}^{-1/2}\boldsymbol{\Sigma}_p^{1/2} \tag{6}$$

Combining everything, we obtain:

$$\nabla_{\boldsymbol{\Sigma}_q} \mathcal{W}_2^2[p, q] = \boldsymbol{I} - \boldsymbol{\Sigma}_p^{1/2}(\boldsymbol{\Sigma}_p^{1/2}\boldsymbol{\Sigma}_q\boldsymbol{\Sigma}_p^{1/2})^{-1/2}\boldsymbol{\Sigma}_p^{1/2} \tag{7}$$

This form may appear complex, but $\boldsymbol{\Sigma}_p^{1/2}$ can be precomputed. It suffices to compute the square root and inverse square root of the symmetric positive definite matrix $\boldsymbol{S} = \boldsymbol{\Sigma}_p^{1/2}\boldsymbol{\Sigma}_q\boldsymbol{\Sigma}_p^{1/2}$, which can be done using the `eigh` function, or via the Newton-Schulz iteration method described in *Efficient Computation of Matrix Square Root and Inverse Square Root* and *Efficient Computation of Matrix r-th Root and Inverse r-th Root*.

4 Large Batches

Introduce the notation:

$$\begin{aligned} \boldsymbol{\mu}_p &= \mathbb{E}[\boldsymbol{z}_p], \qquad \boldsymbol{V}_p = \mathbb{E}[\boldsymbol{z}_p\boldsymbol{z}_p^\top], \qquad \boldsymbol{z}_p = \phi(\boldsymbol{x}_p), \qquad \boldsymbol{x}_p \sim p \\ \boldsymbol{\mu}_q &= \mathbb{E}[\boldsymbol{z}_q], \qquad \boldsymbol{V}_q = \mathbb{E}[\boldsymbol{z}_q\boldsymbol{z}_q^\top], \qquad \boldsymbol{z}_q = \phi(\boldsymbol{x}_q), \qquad \boldsymbol{x}_q \sim q \end{aligned} \tag{8}$$

Then:

$$\boldsymbol{\Sigma}_p = \boldsymbol{V}_p - \boldsymbol{\mu}_p\boldsymbol{\mu}_p^\top, \qquad \boldsymbol{\Sigma}_q = \boldsymbol{V}_q - \boldsymbol{\mu}_q\boldsymbol{\mu}_q^\top \tag{9}$$

Note that $\boldsymbol{z} = \phi(\boldsymbol{x})$ typically has thousands of dimensions (e.g., 2048 for InceptionV3), so accurate estimation often requires tens of thousands of samples. The real distribution is fixed and $\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p$ can be precomputed. However, the generated distribution changes dynamically, and computing accurate estimates every step with tens of thousands of samples implies an enormous batch size, which is computationally prohibitive in many settings.

On the other hand, the gradient formula (7) also reveals the necessity of large batches. With small batches, the estimated $\boldsymbol{V}_q$ may not be full rank, so $\boldsymbol{\Sigma}_q$ may not be invertible, making $(\boldsymbol{\Sigma}_p^{1/2}\boldsymbol{\Sigma}_q\boldsymbol{\Sigma}_p^{1/2})^{-1/2}$ undefined (due to division by zero). Therefore, using FD as a loss imposes a requirement on the training batch size—this is likely the most critical practical difficulty.

Given computational constraints, we aim to simulate large batch effects using small batches, similar to the "contrastive learning + gradient accumulation" scenario.

5 Equivalent Loss

Suppose a batch size B is required for accurate estimation of $\boldsymbol{\mu}_q, \boldsymbol{V}_q$. Instead, we process $k = B/b$ small batches of size b with intermediate results $\tilde{\boldsymbol{\mu}}_q^{(1)}, \tilde{\boldsymbol{V}}_q^{(1)}$, $\tilde{\boldsymbol{\mu}}_q^{(2)}, \tilde{\boldsymbol{V}}_q^{(2)}$, ..., $\tilde{\boldsymbol{\mu}}_q^{(k)}, \tilde{\boldsymbol{V}}_q^{(k)}$, such that:

$$\boldsymbol{\mu}_q = \frac{1}{k}\sum_{i=1}^{k} \tilde{\boldsymbol{\mu}}_q^{(i)}, \qquad \boldsymbol{V}_q = \frac{1}{k}\sum_{i=1}^{k} \tilde{\boldsymbol{V}}_q^{(i)} \tag{10}$$

We seek an equivalent total gradient equal to the sum of gradients over small batches, achieving unbiased estimation. Differentiating both sides of (1):

$$\begin{aligned} d\mathcal{F}(\boldsymbol{\mu}_q, \boldsymbol{V}_q) &= \langle \nabla_{\boldsymbol{\mu}_q}\mathcal{F}, d\boldsymbol{\mu}_q\rangle + \langle \nabla_{\boldsymbol{V}_q}\mathcal{F}, d\boldsymbol{V}_q\rangle_F \\ &= \sum_{i=1}^{k}\left[\langle \nabla_{\boldsymbol{\mu}_q}\mathcal{F}, d\tilde{\boldsymbol{\mu}}_q^{(i)}/k\rangle + \langle \nabla_{\boldsymbol{V}_q}\mathcal{F}, d\tilde{\boldsymbol{V}}_q^{(i)}/k\rangle_F\right] \\ &= d\sum_{i=1}^{k}\mathcal{F}\left(\left[\boldsymbol{\mu}_q - \tilde{\boldsymbol{\mu}}_q^{(i)}/k\right]_{\text{sg}} + \tilde{\boldsymbol{\mu}}_q^{(i)}/k, \left[\boldsymbol{V}_q - \tilde{\boldsymbol{V}}_q^{(i)}/k\right]_{\text{sg}} + \tilde{\boldsymbol{V}}_q^{(i)}/k\right) \end{aligned} \tag{11}$$

This means we compute $\tilde{\boldsymbol{\mu}}_q^{(i)}, \tilde{\boldsymbol{V}}_q^{(i)}$ in small batches, average them to obtain an accurate $\boldsymbol{\mu}_q, \boldsymbol{V}_q$, then compute gradients using the loss:

$$\mathcal{F}_i = \mathcal{F}\left(\left[\boldsymbol{\mu}_q - \tilde{\boldsymbol{\mu}}_q^{(i)}/k\right]_{\text{sg}} + \tilde{\boldsymbol{\mu}}_q^{(i)}/k, \left[\boldsymbol{V}_q - \tilde{\boldsymbol{V}}_q^{(i)}/k\right]_{\text{sg}} + \tilde{\boldsymbol{V}}_q^{(i)}/k\right) \tag{12}$$

and sum the gradients. This yields gradients equivalent to using a batch size B. Here, $[\cdot]_{\text{sg}}$ denotes the stop-gradient operator. Alternatively, we can update parameters each step with a smaller learning rate, achieving a similar effect.

6 Historical Averaging

The above approach is theoretically sound, but not smooth: k forward passes are needed before computing accurate $\boldsymbol{\mu}_q, \boldsymbol{V}_q$, and then gradients are computed retroactively. The bottleneck is the need for global statistics to compute unbiased local gradients.

A natural idea: can we approximate $\boldsymbol{\mu}_q, \boldsymbol{V}_q$? Given small learning rates, model parameters change slowly, so $\boldsymbol{\mu}_q, \boldsymbol{V}_q$ should also change slowly. After including a new batch, the updated statistic is only a small adjustment to the previous one. We use exponential moving averages (EMA):

$$\boldsymbol{\mu}_q^{(t)} = \beta \boldsymbol{\mu}_q^{(t-1)} + (1-\beta)\tilde{\boldsymbol{\mu}}_q^{(t)}, \qquad \boldsymbol{V}_q^{(t)} = \beta \boldsymbol{V}_q^{(t-1)} + (1-\beta)\tilde{\boldsymbol{V}}_q^{(t)} \tag{13}$$

This maintains an effective averaging window of size $\mathcal{O}(1/(1-\beta))$, effectively increasing the statistical batch size by a factor of $\mathcal{O}(1/(1-\beta))$. Thus, at each step we can compute the gradient using the loss:

$$\mathcal{F}_t = \mathcal{F}\left(\underbrace{\beta\big[\boldsymbol{\mu}_q^{(t-1)}\big]_{\mathrm{sg}} + (1-\beta)\tilde{\boldsymbol{\mu}}_q^{(t)}}_{\boldsymbol{\mu}_q^{(t)}}, \underbrace{\beta\big[\boldsymbol{V}_q^{(t-1)}\big]_{\mathrm{sg}} + (1-\beta)\tilde{\boldsymbol{V}}_q^{(t)}}_{\boldsymbol{V}_q^{(t)}}\right) \tag{14}$$

Extra cost: storing $\boldsymbol{\mu}_q, \boldsymbol{V}_q$, which is minimal. This practice of "making up for small batches with historical data" embodies the "streaming" idea in Streaming Power Iteration. Additionally, the paper discusses a queue-based method that retains k historical batches, computes gradients using (12) and removes the oldest batch. This method is simpler but more memory-intensive than EMA and performs worse in practice.

7 Experimental Overview

The paper's experiments focus on post-training of generative models, aiming to improve one-step generation via FD Loss, or fine-tune multi-step models to one-step. When combining multiple encoders to compute FD Loss, a loss normalization technique balances losses of different scales:

$$\mathcal{L} = \sum_i \frac{\mathcal{F}[\phi_i]}{\big[\mathcal{F}[\phi_i]\big]_{\mathrm{sg}} + \epsilon} \tag{15}$$

This technique was also discussed in *On Multi-Task Learning (I): The Name of the Loss*.

The paper's key achievement is pushing one-step generation performance (FID) to unprecedented levels, surpassing both one-step and multi-step baselines. This result appears near the performance ceiling. Some figures are shown below:

Table 2: ***FD-loss* repurposes multi-step JiT models to generate in one step.** All post-trained models use 1 NFE. Setting: JiT-L/16 [23], post-trained for 50 epochs. †200 NFE = 50 steps × 2 (Heun) × 2 (CFG).

setting	NFE	FID↓	IS↑	FDr^6 ↓
base model				
JiT-L (50-step)	200†	2.59	288.5	10.73
JiT-L (1-step)	1	291.59	2.0	214.75
models post-trained with FD-loss				
FD-Incep.	1	**0.77**	293.7	12.86
FD-MAE	1	6.52	280.4	9.30
FD-SigLIP	1	5.10	329.6	9.04
FD-SigLIP+MAE	1	4.67	**354.0**	3.83
FD-SigLIP+Incep.+MAE (**SIM**)	1	0.85	319.5	**3.29**

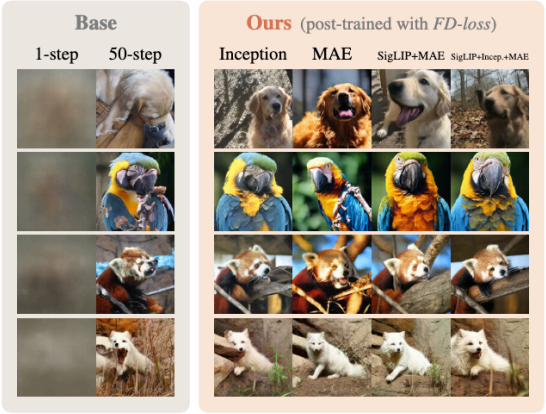

Figure 5: **Repurposing a multi-step model into a one-step generator with *FD-loss*.** Samples from the *same* noise input across the base model and different post-trained models. The naive one-step base model fails to produce sensible images. After post-training, the 1-NFE models generate sensible images, and the strongest variants are visually comparable or superior to the 50-step base model.

Figure 1: FD loss improves one-step generation performance

8 Summary

This article theoretically analyzes the challenges of using FID as a loss function for generative models and derives corresponding techniques to overcome them.

Table 4: **System-level comparison on ImageNet** 256×256**.** All metrics for all methods are computed by us under a unified evaluation pipeline; numbers may differ slightly from the original papers. Our *FD-loss* (shaded rows) improves already strong generators across geneartor families and model scales. †CFG is applied only in a time sub-interval; we report the full-CFG upper bound for simplicity. Uncurated qualitative samples are in Appendix E.

method	NFE	space	#params	FDr^6 ↓	FID↓	IS↑	Prec↑	Recall↑
reference (real images)								
50k validation images	N/A	N/A	N/A	1.00	1.68	232.2	0.75	0.66
discrete-space models								
VAR-d30	10×2	discrete	2B	6.70	1.97	**304.6**	0.82	0.59
BAR-L [57]	256×2×4	discrete	1.1B	**3.57**	**1.01**	281.9	0.77	0.68
latent-space models, multi-step								
without semantic distillation								
SiT-XL/2 [33]	250×2	latent	675M	8.44	2.12	256.7	0.81	0.60
MAR-L [24]	256×2×100	latent	478M	6.68	1.80	293.4	0.80	0.60
FlowAR-H [41]	50×2†	latent	1.9B	6.13	1.68	274.1	0.80	0.62
MAR-H [24]	256×2×100	latent	942M	5.61	1.56	299.5	0.80	0.62
MAR-L, DeTok [54]	256×2×100	latent	478M	**5.49**	**1.39**	**306.2**	0.81	0.62
with semantic distillation								
REG [51]	250×2†	latent	685M	4.64	1.54	302.9	0.78	0.62
SiT-XL/2-REPA [58]	250×2†	latent	675M	5.45	1.42	306.1	0.80	0.65
LightningDiT [55]	250×2	latent	675M	4.57	1.42	294.3	0.80	0.64
DDT-XL [50]	250×2	latent	675M	5.70	1.26	**309.3**	0.79	0.66
REPA-E [22]	250×2†	latent	676M	**3.04**	1.17	298.3	0.79	0.66
RAE-XL [59]	50×2†	latent	839M	3.26	**1.16**	261.0	0.77	0.67
latent-space models, one-step								
Drift-L (latent) [6]	1	latent	463M	10.92	1.53	257.2	0.79	0.63
iMF-XL [12]	1	latent	610M	8.39	1.82	278.9	0.78	0.63
iMF-XL [12]	2	latent	610M	7.48	1.61	289.1	0.79	0.63
+ *FD-loss*	1	latent	610M	**2.45**	**0.76**	301.3	0.77	0.67
pixel-space models, multi-step								
PixNerd-XL [49]	100×2	pixel	1.0B	5.01	2.10	**318.8**	0.81	0.59
JiT-L [23]	50×2×2†	pixel	459M	10.73	2.59	288.5	0.79	0.59
+ *FD-loss*	1	pixel	459M	3.24	0.77	317.3	0.77	0.66
JiT-H [23]	50×2×2†	pixel	953M	7.66	1.97	296.0	0.78	0.63
+ *FD-loss*	1	pixel	953M	**2.65**	**0.75**	313.0	0.76	0.66
pixel-space models, one-step								
Drift-L (pixel) [6]	1	pixel	465M	10.51	1.43	305.8	0.81	0.60
pMF-L [30]	1	pixel	410M	9.09	2.72	261.7	0.81	0.56
+ *FD-loss*	1	pixel	410M	2.09	0.78	309.2	0.76	0.67
pMF-H [30]	1	pixel	935M	6.87	2.29	267.2	0.80	0.59
+ *FD-loss*	1	pixel	935M	**1.89**	0.77	**310.1**	0.77	0.68

Figure 2: Comprehensive comparison of FD loss