Input Perturbation Reduces Exposure Bias in Diffusion Models

Mang Ning [1 2] **Enver Sangineto** [2] **Angelo Porrello** [2] **Simone Calderara** [2] **Rita Cucchiara** [2]

Abstract

Denoising Diffusion Probabilistic Models have shown an impressive generation quality although their long sampling chain leads to high computational costs. In this paper, we observe that a long sampling chain also leads to an error accumulation phenomenon, which is similar to the *exposure bias* problem in autoregressive text generation. Specifically, we note that there is a discrepancy between training and testing, since the former is conditioned on the ground truth samples, while the latter is conditioned on the previously generated results. To alleviate this problem, we propose a very simple but effective training regularization, consisting in perturbing the ground truth samples to simulate the inference time prediction errors. We empirically show that, without affecting the recall and precision, the proposed input perturbation leads to a significant improvement in the sample quality while reducing both the training and the inference times. For instance, on CelebA 64×64, we achieve a new state-of-the-art FID score of 1.27, while saving 37.5% of the training time. The code is available at `https://github.com/forever208/DDPM-IP`.

1. Introduction

Denoising Diffusion Probabilistic Models (DDPMs) (Sohl-Dickstein et al., 2015; Ho et al., 2020) are a new generative paradigm which is attracting a growing interest due to its very high-quality sample generation capabilities (Dhariwal & Nichol, 2021; Nichol et al., 2022; Ramesh et al., 2022). Differently from most existing generative methods which synthesize a new sample in a single step, DDPMs resemble the Langevin dynamics (Welling & Teh, 2011) and the generation process is based on a sequence of denoising steps, in which a synthetic sample is created starting from pure noise and autoregressively reducing the noise component. In more detail, during training, a real sample $\boldsymbol{x}_0$ is progressively destroyed in T steps adding Gaussian noise (*forward process*). The sequence $\boldsymbol{x}_0, ..., \boldsymbol{x}_t, ..., \boldsymbol{x}_T$ so obtained, is used to train a deep denoising autoencoder ($\mu(\cdot)$) to invert the forward process: $\hat{\boldsymbol{x}}_{t-1} = \mu(\boldsymbol{x}_t, t)$. At inference time, the generation process is autoregressive because it depends on the previously generated samples: $\hat{\boldsymbol{x}}_{t-1} = \mu(\hat{\boldsymbol{x}}_t, t)$ (Sec. 3).

Despite the large success of DDPMs in different generative fields (Sec. 2), one of the main drawbacks of these models is their very long computational time, which depends on the large number of steps T required at both the training and the inference stage. As recently emphasised in (Xiao et al., 2022), the fundamental reason why T needs to be large is that each denoising step is assumed to be Gaussian, and this assumption holds only for small step sizes. Conversely, with larger step sizes, the prediction network ($\mu(\cdot)$) needs to solve a harder problem and it becomes progressively less accurate (Xiao et al., 2022). However, in this paper, we observe that there is a second phenomenon, related to the sampling chain, but partially in contrast with the first, which is the *accumulation* of these errors over the T inference sampling steps. This is basically due to the discrepancy between the training and the inference stage, in which the latter generates a sequence of samples based on the results of the previous steps, hence possibly accumulating errors. In fact, at training time, $\mu(\cdot)$ is trained with a ground truth pair $(\boldsymbol{x}_t, \boldsymbol{x}_{t-1})$ and, given $\boldsymbol{x}_t$, it learns to reconstruct $\boldsymbol{x}_{t-1}$ ($\mu(\boldsymbol{x}_t, t)$). However, at inference time, $\mu(\cdot)$ has no access to the "real" $\boldsymbol{x}_t$, and its prediction depends on the previously generated $\hat{\boldsymbol{x}}_t$ ($\mu(\hat{\boldsymbol{x}}_t, t)$). This input mismatch between $\mu(\boldsymbol{x}_t, t)$, used during training, and $\mu(\hat{\boldsymbol{x}}_t, t)$, used during testing, is similar to the *exposure bias* problem (Ranzato et al., 2016; Schmidt, 2019) shared by other autoregressive generative methods. For example, Rennie et al. (2017) argue that training a network to maximize the likelihood of the next ground-truth word given the previous ground-truth word (called "Teacher-Forcing" (Bengio et al., 2015)) results in error accumulation at inference time, since the model has never been exposed to its own predictions.

In this paper, we first empirically analyze this accumulation error phenomenon. For instance, we show that a standard

[1]Department of Information and Computing Science, Utrecht University, the Netherlands. [2]Department of Engineering (DIEF), University of Modena and Reggio Emilia, Italy. Correspondence to: Mang Ning <m.ning@uu.nl>, Enver Sangineto <enver.sangineto@unimore.it>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

DDPM (Dhariwal & Nichol, 2021), trained with T steps, can generate *better* results using a number of inference steps $T' < T$ (Sec. 6.2). A similar phenomenon was also observed by Nichol & Dhariwal (2021), but the authors did not provide an explanation for that. We believe that the reason for this apparently contrasting result is that while, on the one hand, longer chains can better satisfy the Gaussian assumption in the reverse diffusion process, on the other hand, they lead to a larger accumulation of errors.

Second, in order to alleviate the exposure bias problem, we propose a surprisingly simple yet very effective method, which consists in explicitly modelling the prediction error during training. Specifically, at training time, we perturb $\boldsymbol{x}_t$ and we feed $\mu(\cdot)$ with a noisy version of $\boldsymbol{x}_t$, this way simulating the training-inference discrepancy, and forcing the learned network to take into account possible inference-time prediction errors. Note that our perturbation is different from the content-destroying forward process, because the new noise is *not* used in the ground truth prediction *target* (Sec. 5.2). The proposed method is a training regularization which forces the network to smooth its prediction function: to solve the proposed task, two spatially close points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ should lead to similar predictions $\mu(\boldsymbol{x}_1, t)$ and $\mu(\boldsymbol{x}_2, t)$. This regularization approach is similar to Mixup (Zhang et al., 2018) and the Vicinal Risk Minimization (VRM) principle (Chapelle et al., 2000), where a neighborhood around each sample in the training data is defined and then used to perturb that sample keeping fixed its target class label.

Third, we propose alternative solutions to the exposure bias problem for diffusion models, in which, rather than using input perturbation, we obtain a smoother prediction function $\mu(\cdot)$ by explicitly encouraging $\mu(\cdot)$ to be Lipschitz continuous (Sec. 5.4). The rationale behind this is that a Lipschitz continuous function $\mu(\cdot)$ generates small prediction differences between neighbouring points in its domain, leading to a DDPM which is more robust to the inference-time errors.

Finally, we empirically analyse all the proposed solutions and we show that, despite being all effective for improving the final generation quality, input perturbation is both more efficient and more effective than the explicit minimization of the Lipschitz constant in DDPMs (Sec. 6.1). Moreover, directly perturbing the network input at training time has *no additional training overhead* and this solution is very easy to be reproduced and plugged into existing DDPM frameworks: it can be obtained with just two lines of code without any change in the network architecture or the loss function. We call our method *Denoising Diffusion Probabilistic Models with Input Perturbation (DDPM-IP)* and we show that it can significantly improve the generation quality of state-of-the-art DDPMs (Dhariwal & Nichol, 2021; Song et al., 2021a) and speed up the inference-time sampling. For instance, on the CIFAR10 (Krizhevsky et al., 2009), the ImageNet 32×32 (Chrabaszcz et al., 2017), the LSUN 64×64 (Yu et al., 2015) and the FFHQ 128×128 (Karras et al., 2019) datasets, DDPM-IP, with only 80 sampling steps, generates lower FID scores than the state-of-the-art ADM (Dhariwal & Nichol, 2021) with 1,000 steps, corresponding to a more than $12.5\times$ sampling acceleration.

In summary, our contributions are:

- We show that there is an exposure bias problem in DDPMs which has not been investigated so far.
- To alleviate this problem, we propose different regularization methods whose common goal is to smooth the prediction function, and we specifically suggest input perturbation (DDPM-IP) as the best and the simplest of such solutions.
- Using common benchmarks, we show that DDPM-IP can significantly improve the generation quality and drastically speed up both training and inference.

2. Related Work

Diffusion models were introduced by Sohl-Dickstein et al. (2015) and later improved in (Song & Ermon, 2019; Ho et al., 2020; Song et al., 2021b; Nichol & Dhariwal, 2021). More recently, Dhariwal & Nichol (2021) have shown that DDPMs can yield higher-quality images than Generative Adversarial Networks (GANs) (Goodfellow et al., 2014; Brock et al., 2018). Similarly to GANs, the generation process in DDPMs can be both unconditional and conditioned. For instance, GLIDE (Nichol et al., 2022) learns to generate images according to an input textual sentence. Differently from GLIDE, where the diffusion model is defined on the image space, DALL·E-2 (Ramesh et al. (2022)) uses a DDPM to learn a prior distribution on the CLIP (Radford et al., 2021) space. Text-to-image generation is explored also in Stable Diffusion (Rombach et al., 2021) and Imagen (Saharia et al., 2022). Apart from images, DDPMs can also be used with categorical distributions (Hoogeboom et al., 2021; Gu et al., 2021), in an audio domain (Mittal et al., 2021; Chen et al., 2021), in time series forecasting (Rasul et al., 2021) and in other generative tasks (Yang et al., 2022; Croitoru et al., 2022). Differently from previous work, our goal is not to propose an application-specific prediction network, but rather to investigate the training-testing discrepancy of the DDPMs and propose a solution which can be used in different application fields and jointly with different denoising architectures.

Accelerating the DDPM training or reducing the number of sampling steps T (Sec. 1) have been thoroughly investigated due to their practical implications. For instance, Song et al. (2021a) propose Denoising Diffusion Implicit

Models (DDIMs), based on a non-Markovian diffusion process, which can use a number of inference sampling steps smaller than those used at training time, without retraining the network. Salimans & Ho (2022) propose to distil the prediction network into new networks which progressively reduce the number of sampling steps. However, the disadvantage is the need of training multiple networks. Rombach et al. (2021) speed up sampling by splitting the process into a compression stage and a generation stage, and applying the DDPM on the compressed (latent) space. Hoogeboom et al. (2022) present an order-agnostic DDPM, inspired by XLNet (Yang et al., 2019), in which the sequence $\boldsymbol{x}_0, ..., \boldsymbol{x}_T$ is randomly permuted at training time, leading to a partially parallelized sampling process. Chen et al. (2021) found that, instead of conditioning the prediction network ($\mu(\cdot)$) on a discrete diffusion step t, it is beneficial to condition $\mu(\cdot)$ on a continuous noise level. Similarly, Kong & Ping (2021) introduce continuous diffusion steps, resulting in a unified framework for fast sampling. In order to use larger size sampling steps and a non-Gaussian reverse process (Sec. 1) Xiao et al. (2022) include an adversarial loss in DDPMs and propose Denoising Diffusion GANs. Karras et al. (2022) suggest using Heun's second-order deterministic sampling method, leading to high quality results and fast sampling. Xu et al. (2022) accelerate the generation process of continuous normalizing flow using a Poisson flow generative model. Our approach is orthogonal to these previous works, and it can potentially be used jointly with most of them.

3. Background

Without loss of generality, we assume an image domain and we focus on DDPMs which define a diffusion process on the input space. Following (Nichol & Dhariwal, 2021; Dhariwal & Nichol, 2021), we assume that each pixel value is linearly scaled into $[-1, 1]$. Given a sample $\boldsymbol{x}_0$ from the data distribution $q(\boldsymbol{x}_0)$ and a prefixed noise schedule $(\beta_1, ..., \beta_T)$, a DDPM defines the forward process as a Markov chain which starts from a real image $\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$ and iteratively adds Gaussian noise for T diffusion steps:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \boldsymbol{I}), \tag{1}$$

$$q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0) = \prod_{t=1}^{T} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}), \tag{2}$$

until obtaining a completely noisy image $\boldsymbol{x}_T \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. On the other hand, the *reverse process* is defined by transition probabilities parameterized by $\boldsymbol{\theta}$:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_{t-1}; \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t), \sigma_t \boldsymbol{I}), \tag{3}$$

where $\sigma_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$ with $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$ and $\alpha_i = 1 - \beta_i$. Given $\boldsymbol{x}_0$, $\boldsymbol{x}_t$ can be obtained (Ho et al., 2020) by:

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \tag{4}$$

where $\boldsymbol{\epsilon}$ is a noise vector ($\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$). Instead of predicting the mean of the forward process posterior (i.e., $\hat{\boldsymbol{x}}_{t-1} = \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$), Ho et al. (2020) propose to use a network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\cdot)$ which predicts the noise vector ($\boldsymbol{\epsilon}$). Using $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\cdot)$ and a simple L_2 loss function, the training objective becomes:

$$L(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0), \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), t \sim \mathbb{U}(\{1,...,T\})}[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)||^2]. \tag{5}$$

Note that, in Eq. 5, $\boldsymbol{x}_t$ and $\boldsymbol{\epsilon}$ are ground-truth terms, while $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ is the network prediction. Using Eq. 5, the training and the sampling algorithms are described in Alg. 1-2, respectively.

Algorithm 1 DDPM Standard Training

1: **repeat**
2: $\quad \boldsymbol{x}_0 \sim q(\boldsymbol{x}_0), t \sim \mathbb{U}(\{1, ..., T\}), \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
3: $\quad$ compute $\boldsymbol{x}_t$ using Eq. 4
4: $\quad$ take a gradient descent step on $\nabla_{\boldsymbol{\theta}}||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)||^2$
5: **until** converged

Algorithm 2 DDPM Standard Sampling

1: $\hat{\boldsymbol{x}}_T \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
2: **for** $t := T, ..., 1$ **do**
3: $\quad$ if $t > 1$ then $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, else $\boldsymbol{z} = \boldsymbol{0}$
4: $\quad \hat{\boldsymbol{x}}_{t-1} = \frac{1}{\sqrt{\alpha_t}}(\hat{\boldsymbol{x}}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}}_t, t)) + \sigma_t \boldsymbol{z}$
5: **end for**
6: **return** $\hat{\boldsymbol{x}}_0$

4. Exposure Bias Problem in Diffusion Models

Comparing line 4 of Alg. 1 with line 4 of Alg. 2, we note that the inputs of the prediction network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\cdot)$ are different between the training and the inference phase. Concretely, at training time, standard DDPMs use $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$, where $\boldsymbol{x}_t$ is a *ground truth* sample (Eq. 4). In contrast, at inference time, they use $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}}_t, t))$, where $\hat{\boldsymbol{x}}_t$ is computed based on the output of $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\cdot)$ at the previous sampling step t+1. As mentioned in Sec. 1, this leads to a training-inference discrepancy, which is similar to the exposure bias problem observed, e.g., in text generation models, in which the training generation is conditioned on a ground-truth sentence, while the testing generation is conditioned on the previously generated words (Ranzato et al., 2016; Schmidt, 2019; Rennie et al., 2017; Bengio et al., 2015). In order to quantify the error *accumulation* with respect to the number of inference

sampling steps, we use a simple experiment in which we start from a (randomly selected) real image $\boldsymbol{x}_0$, we compute $\boldsymbol{x}_t$ using Eq. 4, and then apply the reverse process (Alg. 2) *starting from* $\boldsymbol{x}_t$ instead of a random $\boldsymbol{x}_T$. This way, when t is small enough, the network should be able to "recover" the path to $\boldsymbol{x}_0$ (the denoising task is easier). We quantify the total error accumulated in t reverse diffusion steps by comparing the difference between the ground truth distribution $q(\boldsymbol{x}_0)$ and the predicted distribution $q(\hat{\boldsymbol{x}}_0)$ using the FID scores in Tab. 1. The experiment was done using ADM (Dhariwal & Nichol, 2021) (trained with $T = 1,000$) and ImageNet 32×32, and we compute the FID scores using 50k samples. Tab. 1 (first row) shows that *the longer the reverse process, the higher the FID scores*, indicating the existence of an error accumulation which is larger with larger values of t. In Appendix 5, we repeat this experiment using deterministic sampling, which quantifies the error accumulation removing the randomness from the sampling process.

Table 1. An empirical estimate of the exposure bias on ImageNet 32×32.

Model	Number of reverse diffusion steps				
	100	300	500	700	1,000
ADM	0.983	1.808	2.587	3.105	3.544
ADM-IP (ours)	0.972	1.594	2.198	2.539	2.742

Finally, in Tab. 3 we will report the FID scores of ADM on different datasets, which show that most of the best results are obtained in the range from 100 to 300 sampling steps, despite all the models have been trained with 1,000 diffusion steps. These results confirm previous similar observations (Nichol & Dhariwal, 2021), and we believe that the reason for this apparently counterintuitive phenomenon, in which fewer sampling steps lead to a better generation quality, is due to the exposure bias problem. Indeed, while more sampling steps correspond to a reverse process which can be more easily approximated with a Gaussian distribution (Sec. 1), longer sampling trajectories produce a larger accumulation of the prediction errors. Hence, the range $[100, 300]$ leads to a better generation quality because it presumably trades off these two opposing aspects.

5. Method

5.1. Regularization with Input Perturbation

The solution we propose to alleviate the exposure bias problem is very simple: we explicitly model the prediction error using a Gaussian input perturbation at training time. More specifically, we assume that the error of the prediction network in the reverse process at time $t + 1$ is normally distributed with respect to the ground-truth input $\boldsymbol{x}_t$ (see Sec. 5.3). This is simulated using a second, dedicated random noise vector $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, using which, we create a perturbed version ($\boldsymbol{y}_t$) of $\boldsymbol{x}_t$:

$$\boldsymbol{y}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t}(\boldsymbol{\epsilon} + \gamma_t\boldsymbol{\xi}). \tag{6}$$

For simplicity, we use a uniform noise schedule for $\boldsymbol{\xi}$ by setting $\gamma_0 = ... = \gamma_T = \gamma$. In fact, although selecting the best noise schedule $(\beta_1, ..., \beta_T)$ in DDPMs is usually very important to get high-quality results (Ho et al., 2020; Chen et al., 2021), it is nevertheless an expensive hyperparameter tuning operation (Chen et al., 2021). Therefore, to avoid adding a second noise schedule $(\gamma_0, ..., \gamma_T)$ to the training procedure, we opted for a simpler (although most likely suboptimal) solution, in which γ_t does not vary depending on t (more details in Sec. 5.3). In Alg. 3 we show the proposed training algorithm, in which $\boldsymbol{x}_t$ is replaced by $\boldsymbol{y}_t$. In contrast, at inference time, we use Alg. 2 without any change.

Algorithm 3 DDPM-IP: Training with input perturbation

1: **repeat**
2: $\quad\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0), t \sim \mathbb{U}(\{1, ..., T\})$
3: $\quad\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
4: $\quad$compute $\boldsymbol{y}_t$ using Eq. 6
5: $\quad$take a gradient descent step on $\nabla_{\boldsymbol{\theta}}||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}_t, t)||^2$
6: **until** converged

5.2. Discussion

In this section, we analyze the difference between Alg. 3 and Alg. 1. Specifically, in line 5 of Alg. 3, we use $\boldsymbol{y}_t$ as the input of the prediction network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\cdot)$ but we keep using $\boldsymbol{\epsilon}$ as the regression target. In other words, the new noise term ($\boldsymbol{\xi}$) we introduce is used *asymmetrically*, because it is applied to the input but *not* to the prediction target ($\boldsymbol{\epsilon}$). For this reason, Alg. 3 is *not* equivalent to choose a different value of $\boldsymbol{\epsilon}$ in Alg. 1, where $\boldsymbol{\epsilon}$ is instead used *symmetrically* both in the forward process (Eq. 4) and as the target of the prediction network (line 4 of Alg. 1).

This difference is schematically illustrated in Fig. 1, where, for both Alg. 1 (i.e., DDPM) and Alg. 3 (DDPM-IP), we show the corresponding pairs of input and target vectors of the prediction network (respectively, $(\boldsymbol{x}_t, \boldsymbol{\epsilon})$ and $(\boldsymbol{y}_t, \boldsymbol{\epsilon})$). In the same figure, we also show a second version of Alg. 1 (called DDPM-y), where we use the standard training protocol (Alg. 1) but change the noise variance in order to adhere to the same distribution generating $\boldsymbol{y}_t$. In fact, it can be easy shown that $\boldsymbol{y}_t$ in Alg. 3 is generated using the following distribution (see Appendix A.2 for a proof):

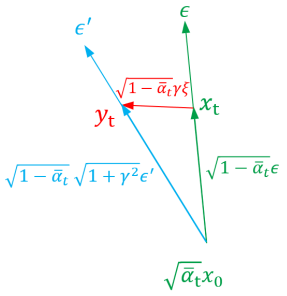
$$q(\boldsymbol{y}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{y}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1 - \bar{\alpha}_t)(1 + \gamma^2)\boldsymbol{I}). \tag{7}$$

Hence, we can obtain *the same input noise distribution* of Alg. 3 in Alg. 1 using $\boldsymbol{\epsilon}' \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and:

$$\boldsymbol{y}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\sqrt{1+\gamma^2}\boldsymbol{\epsilon}'. \tag{8}$$

We call DDPM-y the version of Alg. 1 with this new noise distribution. DDPM-y is obtained from Alg. 1 using Eq. 8 in line 3 and replacing $\boldsymbol{x}_t$ with $\boldsymbol{y}_t$ and $\boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon}'$ in line 4. However, note that, for a given $\boldsymbol{y}_t$, if $\boldsymbol{\xi} \neq \mathbf{0}$, then $\boldsymbol{\epsilon} \neq \boldsymbol{\epsilon}'$ (see Fig. 1), thus, DDPM-IP and DDPM-y share the same input to $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\cdot)$, but they use different targets. In Appendix A.3, we empirically show that DDPM-y is even worse than the standard DDPM.

Intuitively, the proposed training protocol, DDPM-IP, decouples the noise vector $\boldsymbol{\epsilon}'$ actually generating $\boldsymbol{y}_t$ from the ground truth target vector $\boldsymbol{\epsilon}$ which is asked to be predicted by $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\cdot)$. In order to solve this problem, $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\cdot)$ needs to *smooth* its prediction function, reducing the difference between $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ and $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}_t, t)$, and this leads to a training regularization which is similar to VRM (Sec. 1).



	Input	Target
DDPM	x_t	ϵ
DDPM-IP	y_t	ϵ
DDPM-y	y_t	ϵ'

where $\epsilon \sim N(0, I)$ and $\epsilon' \sim N(0, I)$

Figure 1. The inputs and the prediction targets are different in vanilla DDPM, DDPM-IP and DDPM-y.

5.3. Estimating the Prediction Error

In this section, we analyze the actual prediction error of $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\cdot)$ and we use this analysis to choose the value of γ in Eq. 6. Analogously to Sec. 4, we use ADM, *trained using the standard algorithm Alg. 1* and two datasets: CIFAR10 and ImageNet 32×32. At testing time, for a given t and $\hat{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}}_t, t)$, we replace $\boldsymbol{\epsilon}$ with $\hat{\boldsymbol{\epsilon}}$ in Eq. 4 and we compute the predicted $\hat{\boldsymbol{x}}_0$. Finally, the prediction error at time t is $\boldsymbol{e}_t = \hat{\boldsymbol{x}}_0 - \boldsymbol{x}_0$. Note that using $\hat{\boldsymbol{x}}_0$ and $\boldsymbol{x}_0$ to estimate the error instead of comparing $\hat{\boldsymbol{x}}_t$ and $\boldsymbol{x}_t$, has the advantage that the former is independent of scaling factors ($\sqrt{1-\bar{\alpha}_t}$) and, thus, it makes the statistical analysis easier. Using different values of t, uniformly selected in $\{1, ..., T\}$, we empirically verified that, for a given t, $\boldsymbol{e}_t$ is normally distributed: $\boldsymbol{e}_t \sim \mathcal{N}(\mathbf{0}, \nu_t^2\boldsymbol{I})$, with standard deviation ν_t (see Appendix A.5).

In Fig. 2 we plot the value of ν_t with respect to t. The two curves corresponding to the two datasets are surprisingly close to each other. In principle, we could use this empirical analysis and set $\gamma_t = \nu_t$ in Eq. 6. In this way, when we perturb the input to $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\cdot)$, we empirically imitate its actual prediction error which is the base of the exposure bias problem. However, this choice would require a two-step training: first, using Alg. 1 to train the base model and empirically estimate ν_t for different t. Then, using Alg. 3 with the estimated γ_t schedule to retrain the model from scratch. To avoid this and make the whole procedure as simple as possible, we simply use a constant value γ, independently of t. This value was empirically set using a grid search on both CIFAR10 and ImageNet 32×32 on a small range of values covering the last half of the sampling trajectory. Specifically, we investigated the range $\nu_t \in [0, \mathbb{E}_t[\nu_t]] = [0, 0.2]$ (see Fig. 2), which was chosen following Karras et al. (2022), who showed that the last part of the inference trajectory has usually the largest impact on the Diffusion Model performance. We finally set $\gamma = 0.1$ and, in the rest of this paper, we *always* use a constant $\gamma = 0.1$, *regardless of the dataset and the baseline DDPM*. Although a DDPM-specific γ value would most likely lead to better quality results, we prefer to emphasise the ease of use of our proposal *which does not depend on any other hyperparameter*.

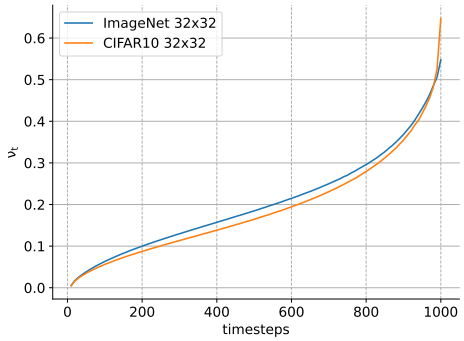


Figure 2. The inference time standard deviation ν_t of the prediction error of a pre-trained network with respect to the sampling step t. The mean of the blue and the orange curve is 0.20 and 0.19, respectively.

5.4. Regularization based on Lipschitz Continuous Functions

In this section, we propose two alternative solutions to the exposure bias problem which can help to better investigate the phenomenon. The goal is the same as in Sec. 5.1, i.e., we want to smooth the prediction function $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ to make it more robust with respect to local variations of $\boldsymbol{x}_t$ which are due to the inference-time prediction errors. To do so, instead of using input perturbation, we explicitly encourage $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\cdot)$ to be Lipschitz continuous, i.e. to satisfy:

$$||\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}, t) - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}, t)|| \leq K||\boldsymbol{x} - \boldsymbol{y}||, \quad \forall(\boldsymbol{x}, \boldsymbol{y}) \tag{9}$$

for a small constant K. We implement this idea using two standard Lipschitz constant minimization methods: *gradient penalty* (Rifai et al., 2011; Gulrajani et al., 2017) and *weight decay* (Krogh & Hertz, 1991; Miyato et al., 2018). In both cases we do *not* perturb the input of $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\cdot)$, and we use the original training algorithm (Alg. 1), with the only difference being the loss function used in line 4, where the L_2 loss is used jointly with a regularization term described below.

Gradient penalty. In this case, the regularization is based on the Frobenius norm of the Jacobian matrix (Rifai et al., 2011; Goodfellow et al., 2016), and the final loss is:

$$L_{GP}(\boldsymbol{\theta}) = ||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)||^2 + \lambda_{GP} \left\| \frac{\partial \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)}{\partial \boldsymbol{x}} \right\|_F^2, \tag{10}$$

where λ_{GP} is the weight of the gradient penalty term. However, a gradient penalty regularization is very slow (Yoshida & Miyato, 2017) because it involves one forward and two backward passes for each training step.

Weight decay. As shown in (Liu et al., 2022), Lipschitz continuity can also be encouraged using a weight decay regularization (see Appendix A.6 for more details). In this case, the final loss is:

$$L_{WD}(\boldsymbol{\theta}) = ||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)||^2 + \lambda_{WD}||\boldsymbol{\theta}||^2, \tag{11}$$

where λ_{WD} is the weight of the regularization term.

6. Results

In this section, we evaluate the generation quality of the proposed solutions and we compare them with state-of-the-art DDPMs. We use unconditional image generation tasks on different datasets and standard metrics: the Fréchet Inception Distance (FID) (Heusel et al., 2017) and the Spatial Fréchet Inception Distance (sFID) (Nash et al., 2021). As a variant of FID, sFID uses spatial features rather than the standard pooled features to better capture spatial relationships, rewarding image distributions with a coherent high-level structure. As mentioned in Sec. 5.3, in **all** our experiments we use $\gamma = 0.1$ *without any dataset or baseline specific tuning of our only hyperparameter*.

6.1. Evaluation of the Different Proposed Solutions

In this section, we empirically compare to each other the three regularization methods proposed in Sec. 5 to alleviate the exposure bias problem. For all three approaches, we use the state-of-the-art diffusion model ADM (Dhariwal & Nichol, 2021) (without classifier guidance) as the baseline, and we call: (1) "ADM-IP" the version of ADM trained using Alg. 3, (2) "ADM-GP" the version of ADM trained using the gradient penalty, and (3) "ADM-WD" for the weight decay (Sec. 5.4). We use $\lambda_{GP} = 1e{-}6$ and $\lambda_{WD} = 0.03$ as the loss weights for ADM-GP and ADM-WD, respectively.

For this experiment, we use CIFAR10 because ADM-GP is too time-consuming to be trained on larger datasets. The results in Tab. 2 show that all three models outperform the baseline in image quality, demonstrating the effectiveness of smoothing the prediction function using the proposed regularization methods. However, training ADM-GP is too slow and cannot be scaled to larger datasets, thus we do not recommend this solution. Moreover, ADM-IP gets the best FID and sFID scores, thus, in the rest of this paper, we use the input perturbation approach described in Sec. 5.1 as our basic solution.

Table 2. Comparison of different regularization methods. All the models are tested using $T = 1,000$ sampling steps.

Model	CIFAR10 32×32	
	FID	sFID
ADM (baseline)	2.99	4.76
ADM-GP	2.80	4.41
ADM-WD	2.82	4.61
ADM-IP	**2.76**	**4.05**

Finally, we use ADM-IP to quantify the reduction in the exposure bias following the protocol described in Sec. 4. The results reported in Tab. 1 show that ADM-IP leads to a significantly lower exposure bias than ADM, and this difference is larger with longer sampling sequences.

6.2. Main results

Comparison with DDPMs. We compare ADM-IP with ADM using CIFAR10, ImageNet 32×32, LSUN tower 64×64, CelebA 64×64 (Liu et al., 2015) and FFHQ 128×128. Following prior work (Ho et al., 2020; Nichol & Dhariwal, 2021), we generate 50K samples for each trained model and we use the full training set to compute the reference distribution statistics, except for LSUN tower where (again following (Ho et al., 2020; Nichol & Dhariwal, 2021)) we use 50K training samples as the reference data. When training, we always use $T = 1,000$ steps for all the models. At inference time, the results reported with $T' < T$ sampling steps have been obtained using the *respacing* technique (Nichol & Dhariwal, 2021). As previously mentioned (see Sec. 5.3) we keep fixed $\gamma = 0.1$ in all the experiments and the datasets. We refer to Appendix A.7 for the complete list of hyperparameters (e.g. the learning rate, the batch size,

Table 3. Comparison between ADM and ADM-IP using models trained with $T = 1,000$ sampling steps and tested with $T' \leq T$ steps.

Sampling steps (T')	Model	CIFAR10		ImageNet 32		LSUN tower 64		CelebA 64		FFHQ 128	
		FID	sFID	FID	sFID	FID	sFID	FID	sFID	FID	sFID
1,000	ADM (baseline)	2.99	4.76	3.60	3.30	3.39	7.96	1.60	3.80	9.65	12.53
	ADM-IP (ours)	**2.76**	**4.05**	**2.87**	**2.39**	**2.68**	**6.04**	**1.31**	**3.38**	**2.98**	**5.59**
300	ADM	2.95	4.95	3.58	3.48	3.31	8.39	1.82	4.25	9.55	12.6
	ADM-IP	**2.67**	**4.14**	**2.74**	**2.58**	**2.60**	**5.98**	**1.43**	**3.36**	**3.74**	**5.97**
100	ADM	3.37	5.66	4.26	4.48	3.50	11.10	3.02	5.76	14.52	16.02
	ADM-IP	**2.70**	**4.51**	**3.24**	**3.13**	**2.79**	**6.56**	**2.21**	**4.33**	**5.94**	**7.90**
80	ADM	3.63	5.97	4.61	4.76	4.17	12.60	3.75	6.80	17.00	18.02
	ADM-IP	**2.93**	**4.69**	**3.57**	**3.33**	**2.95**	**6.93**	**2.67**	**4.69**	**6.89**	**8.79**

etc.) and network architecture settings, which are the same for both ADM and ADM-IP.

The results reported in Tab. 3 show that, independently of the dataset and the number of sampling steps ($T' \leq T$), ADM-IP is *always* better than ADM in terms of both the FID and sFID metrics, sometimes drastically better. For instance, on LSUN, with $T' = 80$, we have a more than 5 sFID score improvement with respect to ADM. On FFHQ 128×128, with $T' = 1,000$, we have almost 7 points of improvement compared to both the FID and the sFID scores. In addition to the experiments shown in Tab. 3, we used $T' = 900$ sampling steps and our ADM-IP on CelebA 64×64, achieving a result of 1.27 FID, which is the new state-of-the-art performance for unconditional generation on this dataset.

Note that, for most datasets, both the baseline (ADM) and ADM-IP reach the best results with $T' < T$ (specifically, with $T' \in [100, 300]$). As mentioned in Sec. 4, this is most likely a confirmation of the exposure bias problem: a shorter sampling trajectory accumulates a smaller prediction error.

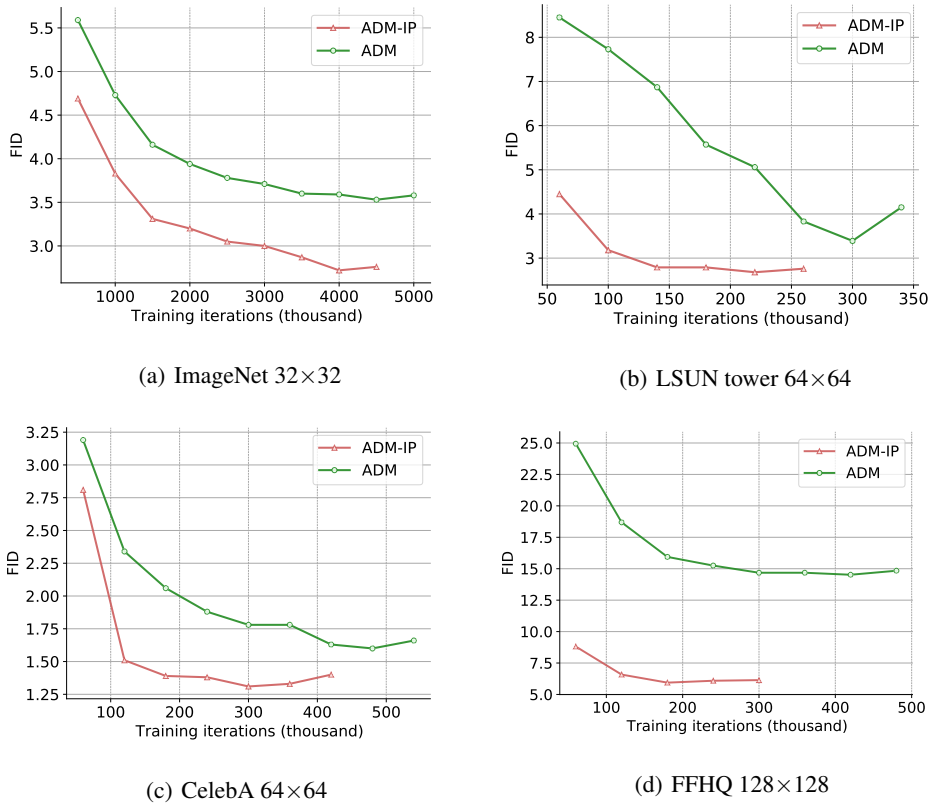
Besides generating significantly better images, ADM-IP converges much faster than the baseline during training in all the five datasets (see Fig. 3 and 4). For instance, on LSUN tower and CelebA, ADM-IP converges at 220K and 300K training iterations while ADM saturates around 300K and 480K iterations, respectively. Fig. 3 shows also that, even before convergence, ADM-IP quickly beats the ADM results obtained when the latter has converged. For instance, on CelebA, ADM-IP gets FID 1.51 at 120K training iterations, whereas ADM gets FID 1.6 at convergence (480K iterations), exhibiting a 4x training speed-up. On the larger resolution FFHQ dataset, ADM receives FID 14.52 at convergence (420K iterations), while ADM-IP achieves a FID score of 8.81 with only 60K iterations: an improvement of 5.71 points with a 7x training speed-up. Fig. 4 shows a similar trend for the CIFAR10 dataset. In this figure, we also plot the results of ADM-IP with different γ values (Sec. 5.3).

The training iterations until convergence for each model are summarized in Tab. 4. The much faster convergence of our method is most likely due to the regularization effect of the input perturbation. In fact, as commonly happens with regularization techniques (Zhang et al., 2018; Liu et al., 2021; Balestriero et al., 2022), the proposed input perturbation also introduces an inductive bias in training. In our case, it is: close points in the domain of the prediction function should lead to similar outcomes. Our empirical results show that this bias helps the DDPM training.

Tab. 4 also shows that ADM-IP can drastically accelerate the inference process, i.e. obtaining better results than the baseline with shorter sampling trajectories. For example, with only 60 or 80 steps, ADM-IP gets a better or an equivalent FID than ADM (tested with the standard 1,000 sampling steps) on all datasets, except for CelebA, where ADM-IP needs 200 sampling steps to reach the same result. This comparison shows a remarkable 5x to 16.7x speed-up of the inference stage, which is particularly significant for the larger resolution FFHQ dataset.

Finally, we measure the recall and precision for the generated samples using the method in Kynkäänniemi et al. (2019). The results show that the recall and precision achieved by ADM and ADM-IP have no significant difference, which indicates that our input perturbation does not affect the sample diversity (see Appendix A.4).

Comparison with DDIMs. In order to show the generality of our proposal, we use Alg. 3 with the Denoising Diffusion Implicit Models (DDIMs) proposed by Song et al. (2021a) (Sec. 2). We train both the baseline (DDIM) and our method (DDIM-IP) on CIFAR10 using the public code provided by Song et al. (2021a). Since training with DDIM is particularly slow, we use only CIFAR10 for this comparison. We use the default hyperparameters settings (e.g. $T = 1,000$)



(a) ImageNet 32×32

(b) LSUN tower 64×64

(c) CelebA 64×64

(d) FFHQ 128×128

Figure 3. FID scores with respect to the number of training iterations. Each FID value is computed using $T' = 1,000$ inference sampling steps, except for the FFHQ dataset, for which we used $T' = 100$.

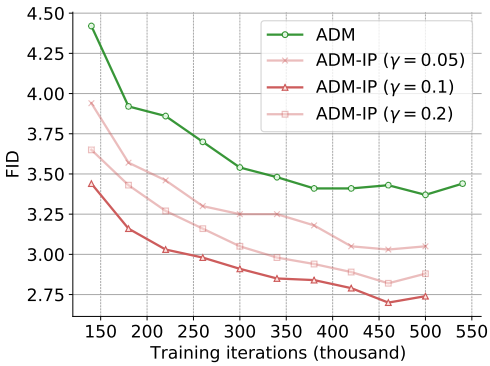


Figure 4. CIFAR10: FID scores with respect to the number of training iterations with different γ values. Each FID score is computed using $T' = 100$ inference sampling steps.

in their code and train both models for 1,600K iterations with batch size 128. We test the performance of the two models with both $\eta = 0$ and $\eta = 0.5$, where η is the coefficient of stochasticity sampling in DDIMs. Also in this case, for our method (DDIM-IP) we use $\gamma = 0.1$ *without any fine-tuning*.

We report the results in Tab. 5, which show that DDIM-IP consistently obtains better FID scores than DDIM in *all* conditions (i.e., independently of the number of sampling steps and the value of η). Importantly, the fewer the sampling steps, the more the FID gain which is obtained with input perturbation. For instance, with $\eta = 0.5$, the FID gain of DDIM-IP is 7.16 with 10 sampling steps versus 0.89 with 1,000 sampling steps. Analogously, with $\eta = 0$ and 10 sampling steps, DDIM-IP drastically improves DDIM with a 3.67 FID margin. Since the main advantage of DDIMs with respect to DDPMs is their reduced number of sampling steps (Song et al., 2021a), and they indeed are mainly used for accelerating the inference stage, input perturbation greatly matches this goal, and it significantly improves the sample quality of the implicit models in a short sampling sequence regime.

7. Conclusions

In this paper, we proposed DDPM-IP, a regularization method for DDPM training which is based on input perturbation to explicitly model the prediction errors and alleviate

Table 4. ADM-IP training and testing acceleration. Note that, for a single training iteration, ADM and ADM-IP take exactly the same amount of time, and the same is true for a single sampling step.

Dataset	Model	Training iterations	Sampling steps	FID
CIFAR10	ADM	500K	1,000	2.99
32×32	ADM-IP	460K	80	2.93
ImageNet	ADM	4500K	1,000	3.53
32×32	ADM-IP	4000K	80	3.50
LSUN tower	ADM	300K	1,000	3.39
64×64	ADM-IP	220K	60	3.31
CelabA	ADM	480K	1,000	1.60
64×64	ADM-IP	300K	200	1.53
FFHQ	ADM	420K	1,000	9.65
128×128	ADM-IP	180K	60	8.72

Table 5. CIFAR10: Comparison between DDIM and DDIM-IP using models trained with $T = 1,000$ sampling steps and tested with $T' \leq T$ steps.

η	Model	Sampling steps (T')				
		10	20	50	100	1,000
0	DDIM	14.21	7.50	5.17	4.66	4.29
	DDIM-IP	**10.54**	**5.70**	**4.66**	**4.52**	**4.27**
0.5	DDIM	17.24	8.87	5.59	4.88	4.45
	DDIM-IP	**10.06**	**5.53**	**3.95**	**3.66**	**3.56**

the DDPM exposure bias problem. We empirically showed that DDPM-IP can significantly improve image quality and drastically reduce both the training and the inference time. The proposed method is straightforward and does not require any change in the network architecture or the specific loss function. This simplicity makes it very easy to be reproduced and plugged into existing DDPMs. Although we tested DDPM-IP only on an image domain, there are no domain-specific assumptions behind our method, hence we presume it can be more generally applied to other domains.

Limitations. Since training DDPMs is very computationally heavy, in this paper we used only datasets with small resolution images. We leave the extension of our experiments to larger resolution images (and corresponding larger backbone networks) as a future work. However, we emphasize that our best results have been obtained with FFHQ 128×128, which is the dataset with the largest resolution images we tested, which probably confirms that our regularization method is specifically effective with higher dimensional input spaces.

Acknowledgments

This work has been supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 955778. Moreover, we acknowledge the CINECA award under the ISCRA initiative, for the availability of high-performance computing resources and support.

References

Balestriero, R., Bottou, L., and LeCun, Y. The effects of regularization and data augmentation are class dependent. *arXiv:2204.03632*, 2022.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. In *NeurIPS*, 2015.

Brock, A., Donahue, J., and Simonyan, K. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

Chapelle, O., Weston, J., Bottou, L., and Vapnik, V. Vicinal risk minimization. In *NIPS*, 2000.

Chen, N., Zhang, Y., Zen, H., Weiss, R. J., Norouzi, M., and Chan, W. Wavegrad: Estimating gradients for waveform generation. In *ICLR*, 2021.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv:1707.08819*, 2017.

Croitoru, F.-A., Hondru, V., Ionescu, R. T., and Shah, M. Diffusion models in vision: A survey. *arXiv preprint arXiv:2209.04747*, 2022.

Dhariwal, P. and Nichol, A. Q. Diffusion models beat GANs on image synthesis. In *NeurIPS*, 2021.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *NeurIPS*, 2014.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Gu, S., Chen, D., Bao, J., Wen, F., Zhang, B., Chen, D., Yuan, L., and Guo, B. Vector quantized diffusion model for text-to-image synthesis. *arXiv:2111.14822*, 2021.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. Improved training of wasserstein gans. *NeurIPS*, 30, 2017.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *NeurIPS*, 30, 2017.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

Hoogeboom, E., Nielsen, D., Jaini, P., Forré, P., and Welling, M. Argmax flows and multinomial diffusion: Learning categorical distributions. In *NeurIPS*, 2021.

Hoogeboom, E., Gritsenko, A. A., Bastings, J., Poole, B., van den Berg, R., and Salimans, T. Autoregressive diffusion models. In *ICLR*, 2022.

Karras, T., Laine, S., and Aila, T. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4401–4410, 2019.

Karras, T., Aittala, M., Aila, T., and Laine, S. Elucidating the design space of diffusion-based generative models. In *NeurIPS*, 2022.

Kong, Z. and Ping, W. On fast sampling of diffusion probabilistic models. *arXiv preprint arXiv:2106.00132*, 2021.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Krogh, A. and Hertz, J. A simple weight decay can improve generalization. *NeurIPS*, 4, 1991.

Kynkäänniemi, T., Karras, T., Laine, S., Lehtinen, J., and Aila, T. Improved precision and recall metric for assessing generative models. *Advances in Neural Information Processing Systems*, 32, 2019.

Liu, H.-T. D., Williams, F., Jacobson, A., Fidler, S., and Litany, O. Learning smooth neural functions via lipschitz regularization. *arXiv preprint arXiv:2202.08345*, 2022.

Liu, Y., Sangineto, E., Bi, W., Sebe, N., Lepri, B., and Nadai, M. D. Efficient training of visual transformers with small datasets. *NeurIPS*, 2021.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *ICLR*, 2019.

Luo, C. Understanding diffusion models: A unified perspective. *arXiv preprint arXiv:2208.11970*, 2022.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al. Mixed precision training. *arXiv:1710.03740*, 2017.

Mittal, G., Engel, J. H., Hawthorne, C., and Simon, I. Symbolic music generation with diffusion models. In *Proceedings of the 22nd International Society for Music Information Retrieval Conference, ISMIR*, 2021.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Nash, C., Menick, J., Dieleman, S., and Battaglia, P. W. Generating images with sparse representations. *arXiv:2103.03841*, 2021.

Nichol, A. Q. and Dhariwal, P. Improved denoising diffusion probabilistic models. In *ICML*, 2021.

Nichol, A. Q., Dhariwal, P., Ramesh, A., Shyam, P., Mishkin, P., McGrew, B., Sutskever, I., and Chen, M. GLIDE: towards photorealistic image generation and editing with text-guided diffusion models. In *ICML*, 2022.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 32, 2019.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning Transferable Visual Models From Natural Language Supervision. In *ICML*, 2021.

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. Hierarchical text-conditional image generation with CLIP latents. *arXiv:2204.06125*, 2022.

Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. Sequence level training with recurrent neural networks. In *ICLR*, 2016.

Rasul, K., Seward, C., Schuster, I., and Vollgraf, R. Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting. In *ICML*, 2021.

Rennie, S. J., Marcheret, E., Mroueh, Y., Ross, J., and Goel, V. Self-critical sequence training for image captioning. In *CVPR*, 2017.

Rifai, S., Pascal Vincent, X. M., Glorot, X., and Bengio, Y. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML*, 2011.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models, 2021.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S. K. S., Ayan, B. K., Mahdavi, S. S., Lopes, R. G., et al. Photorealistic text-to-image diffusion

models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022.

Salimans, T. and Ho, J. Progressive distillation for fast sampling of diffusion models. In *ICLR*, 2022.

Schmidt, F. Generalization in generation: A closer look at exposure bias. In *Proceedings of the 3rd Workshop on Neural Generation and Translation@EMNLP-IJCNLP*, 2019.

Shapiro, S. S. and Wilk, M. B. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4): 591–611, 1965.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, 2015.

Song, J., Meng, C., and Ermon, S. Denoising diffusion implicit models. In *ICLR*, 2021a.

Song, Y. and Ermon, S. Generative modeling by estimating gradients of the data distribution. *NeurIPS*, 32, 2019.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *ICLR*, 2021b.

Welling, M. and Teh, Y. W. Bayesian learning via stochastic gradient langevin dynamics. In *ICML*, 2011.

Xiao, Z., Kreis, K., and Vahdat, A. Tackling the generative learning trilemma with denoising diffusion GANs. In *International Conference on Learning Representations (ICLR)*, 2022.

Xu, Y., Liu, Z., Tegmark, M., and Jaakkola, T. Poisson flow generative models. *arXiv preprint arXiv:2209.11178*, 2022.

Yang, L., Zhang, Z., and Hong, S. Diffusion models: A comprehensive survey of methods and applications. *arXiv:2209.00796*, 2022.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J. G., Salakhutdinov, R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, 2019.

Yoshida, Y. and Miyato, T. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941*, 2017.

Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., and Xiao, J. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Zhang, H., Cissé, M., Dauphin, Y. N., and Lopez-Paz, D. mixup: Beyond empirical risk minimization. In *ICLR*, 2018.

A. Appendix

A.1. Exposure Bias Analysis

In this section, we repeat the experiment in Sec. 4 by removing the randomness component of the sampling process in order to isolate the error of the reverse process which is due only to the prediction network. Specifically, we use again ADM (Dhariwal & Nichol, 2021) (trained with $T = 1,000$) and ImageNet 32×32, and we directly measure the difference between a ground truth real image $\boldsymbol{x}_0$ and the predicted $\hat{\boldsymbol{x}}_0$ using a *deterministic sampling*, described in Alg. 4. In more detail, given a real image $\boldsymbol{x}_0$, we first compute $\boldsymbol{x}_t$ by Eq. 4, then we use the *pre-trained* network $\boldsymbol{\epsilon_\theta}$ (trained with the standard algorithm Alg. 1) to run the reverse diffusion for t steps. Note that we adopt the equation in line 4 of Alg. 2 but we remove the stochastic term $\sigma_t \boldsymbol{z}$. Differently from the analogous experiment presented in Sec. 4, this deterministic reverse diffusion process allows the model to target the mode of $\boldsymbol{x}_0$ instead of favouring diversity (Luo, 2022). Finally, we use the average pixel-wise L_1 distance between $\boldsymbol{x}_0$ and $\hat{\boldsymbol{x}}_0$ to estimate the cumulative error computed in the whole trajectory of t steps. Note that, since each pixel is normalized in $[-1, 1]$ (Sec. 3), then this distance is upper bounded by 2.

Algorithm 4 Deterministic measurement of exposure bias

1: Initialize $\delta_t = 0$, $n_t = 0$ $(\forall t \in \{1, ..., T\})$
2: **repeat**
3: $\quad \boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$, $t \sim \mathbb{U}(\{1, ..., T\})$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
4: $\quad$ compute $\boldsymbol{x}_t$ using Eq. 4
5: $\quad$ **for** $\tau := t, ..., 1$ **do**
6: $\quad\quad \hat{\boldsymbol{x}}_{\tau-1} = \frac{1}{\sqrt{\alpha_\tau}}(\hat{\boldsymbol{x}}_\tau - \frac{1-\alpha_\tau}{\sqrt{1-\bar{\alpha}_\tau}}\boldsymbol{\epsilon_\theta}(\hat{\boldsymbol{x}}_\tau, \tau))$
7: $\quad$ **end for**
8: $\quad \delta_t = \delta_t + ||\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0||_1/M$, where M is the number of pixels in $\boldsymbol{x}_0$
9: $\quad n_t = n_t + 1$
10: **until** N iterations
11: if $n_t \neq 0$, then $\bar{\delta}_t = \frac{\delta_t}{n_t}$ $(\forall t \in \{1, ..., T\})$

In Tab. 6, we report the exposure bias measured using $\bar{\delta}_t$ with respect to different trajectory lengths (t). This table shows that the error accumulates greatly as the number of reverse diffusion steps increases. In Fig. 5 we visualize a few pairs of images $(\boldsymbol{x}_0, \hat{\boldsymbol{x}}_0)$ with the corresponding length of the diffusion trajectory (t). These images clearly show how large the error is accumulated with the diffusion chain getting longer.

Table 6. A deterministic estimate of the exposure bias ($\bar{\delta}_t$) with respect to different lengths of the reverse diffusion trajectory. The error is upper bounded by 2.

Model	Number of reverse diffusion steps			
	100	300	600	1,000
ADM	0.0539	0.1074	0.1821	0.8165

A.2. Distribution of the Perturbed Input

In this section, we prove that $\boldsymbol{y}_t$ is Gaussian distributed as described in Eq. 7. Generally speaking, if $A \sim \mathcal{N}(\mu_A, \sigma_A^2)$ and $B \sim \mathcal{N}(\mu_B, \sigma_B^2)$ are two independent Gaussian distributed random variables, then its linear combination $S = aA + bB$ (with a, b two scalars) is also Gaussian distributed:

$$S \sim \mathcal{N}(a\mu_A + b\mu_B, a^2\sigma_A^2 + b^2\sigma_B^2). \tag{12}$$

In our case, we have that, for a given $\boldsymbol{x}_0$, $\boldsymbol{y}_t$ is a linear combination of $\boldsymbol{x}_t$ and $\boldsymbol{\xi}$, which are two independent, Gaussian distributed random variables:

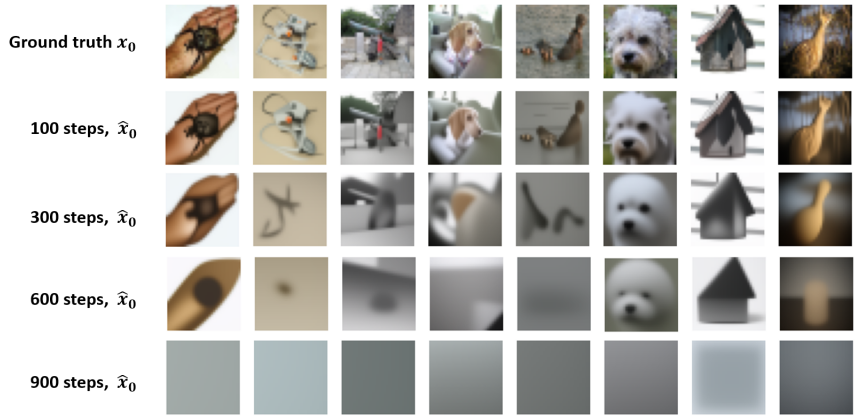


Figure 5. Visualization of the exposure bias problem with different diffusion chain lengths.

$$q(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1-\bar{\alpha}_t)\boldsymbol{I}), \tag{13}$$

$$\boldsymbol{\xi} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{14}$$

$$\boldsymbol{y}_t = \boldsymbol{x}_t + \sqrt{1-\bar{\alpha}_t}\gamma\boldsymbol{\xi}. \tag{15}$$

Hence, if in Eq. 12 we replace S with $\boldsymbol{y}_t$, A with $\boldsymbol{x}_t$, B with $\boldsymbol{\xi}$, and we use $a = 1$ and $b = \sqrt{1-\bar{\alpha}_t}\gamma$, we get:

$$q(\boldsymbol{y}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{y}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1-\bar{\alpha}_t)\boldsymbol{I} + \gamma^2(1-\bar{\alpha}_t)\boldsymbol{I}) = \tag{16}$$

$$= \mathcal{N}(\boldsymbol{y}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1-\bar{\alpha}_t)(1+\gamma^2)\boldsymbol{I}). \tag{17}$$

A.3. Ablation Study: Input Perturbation is not Equivalent to Using a Different Noise Variance

The goal of this section is to empirically show that DDPM-IP is not equivalent to using a standard DDPM algorithm with a different noise distribution. Following the discussion in Sec. 5.2, and adopting the same terminology, we compare DDPM-IP with DDPM-y, where the latter is trained using the standard algorithm (Alg. 1) but adopting the noise distribution of $\boldsymbol{y}_t$. Tab. 7 shows that DDPM-y is even worse than DDPM.

Table 7. CIFAR10: comparing DDPM, DDPM-y and DDPM-IP using different numbers of revers diffusion steps.

Model	Input	Target	80 steps		100 steps		300 steps		1000 steps	
			FID	sFID	FID	sFID	FID	sFID	FID	sFID
DDPM	$\boldsymbol{x}_t$	$\boldsymbol{\epsilon}$	3.63	5.97	3.37	5.66	2.95	4.95	2.99	4.76
DDPM-y	$\boldsymbol{y}_t$	$\boldsymbol{\epsilon}'$	4.24	6.51	3.90	6.23	3.21	5.39	3.25	5.04
DDPM-IP	$\boldsymbol{y}_t$	$\boldsymbol{\epsilon}$	**2.93**	**4.69**	**2.70**	**4.51**	**2.67**	**4.14**	**2.76**	**4.05**

A.4. Recall and Precision

We compare Recall and Precision for ADM and ADM-IP using the improved metrics (Kynkäänniemi et al., 2019) and the code of Dhariwal & Nichol (2021). For each dataset and model, we generate 50,000 samples with 1,000 sampling steps. The results in Tab. 8 indicate that the Recall and Precision values achieved by ADM and ADM-IP have no significant difference, while ADM-IP gets slightly better results on CIFAR10 32×32 and FFHQ 128×128. Note that, due to the limited memory of our NVIDIA V100 16G GPU, we experienced an out-of-memory issue when computing Recall and Precision on the ImageNet 32×32 dataset, thus this result is not reported in Tab. 8.

Table 8. Comparing Recall and Precision for ADM and ADM-IP on the four datasets using 1,000 sampling steps.

Model	CIFAR10 32×32		LSUN tower 64×64		CelebA 64×64		FFHQ 128×128	
	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision
ADM	0.600	0.690	**0.618**	0.631	0.592	**0.703**	0.583	0.690
ADM-IP	**0.606**	**0.696**	0.612	**0.640**	**0.601**	0.700	**0.585**	**0.703**

A.5. Gaussian Prediction Error

In this section, we use ImageNet 32×32 to empirically show that $\boldsymbol{e}_t \sim \mathcal{N}(\mathbf{0}, \nu_t^2 \boldsymbol{I})$ (Sec. 5.3), i.e., that the prediction error is nearly isotropic Gaussian distributed. To do so, we need to prove that, for each t and each input dimension (i.e., for each pixel and color channel) $i \in \{1, ..., M\}$, the pixel-wise error (e_t^i) follows $e_t^i \sim \mathcal{N}(0, \nu_t^2)$. To test this hypothesis, we uniformly select a subset of 100 t values in $\{1, ..., T\}$ using a stride of 10. Then, for each t, we use 10K images and all the pixels to compute the pixel independent mean μ_t and variance ν_t^2 of the error, which we use to standardize the error values for all the pixels e_t^i (i.e., $\bar{e}_t^i = \frac{e_t^i - \mu_t}{\nu_t^2}$). Then, for each i, we use 50 randomly selected $\bar{e}_t^i$ values and the Shapiro–Wilk test (Shapiro & Wilk, 1965) to verify that they follow a standard normal distribution. The confidence level is set at 95% and we reject the null hypothesis if the p-value is less than 0.05. The null hypothesis was rejected only in a small minority of cases, confirming that the error $\boldsymbol{e}_t$ is almost isotropic Gaussian distributed. Fig. 6 shows a few histogram examples for e_t^i computed at different pixels.

A.6. Relation between the Lipschitz Constant Minimization and the Weight Decay Minimization

By definition, in Lipschitz continuos functions, the relation between the output difference $\|f_w(x_1) - f_w(x_2)\|$ and the input difference $\|x_1 - x_2\|$ of two points is governed by a constant K as follows:

$$\|f_w(x_1) - f_w(x_2)\| \leq K \cdot \|x_1 - x_2\| . \tag{18}$$

Since a neural network is usually a stack of layers, without loss of generality we consider a single layer neural network, $f(x) = ReLU(Wx + b)$, thus we have:

$$\|f(Wx_1 + b) - f(Wx_2 + b)\| \leq K \cdot \|x_1 - x_2\| . \tag{19}$$

Using the first order term of Tylor Series to approximate the left side of the above equation, we get:

$$\left\| \frac{\partial f}{\partial y} \cdot W(x_1 - x_2) \right\| \leq K \cdot \|x_1 - x_2\| , \tag{20}$$

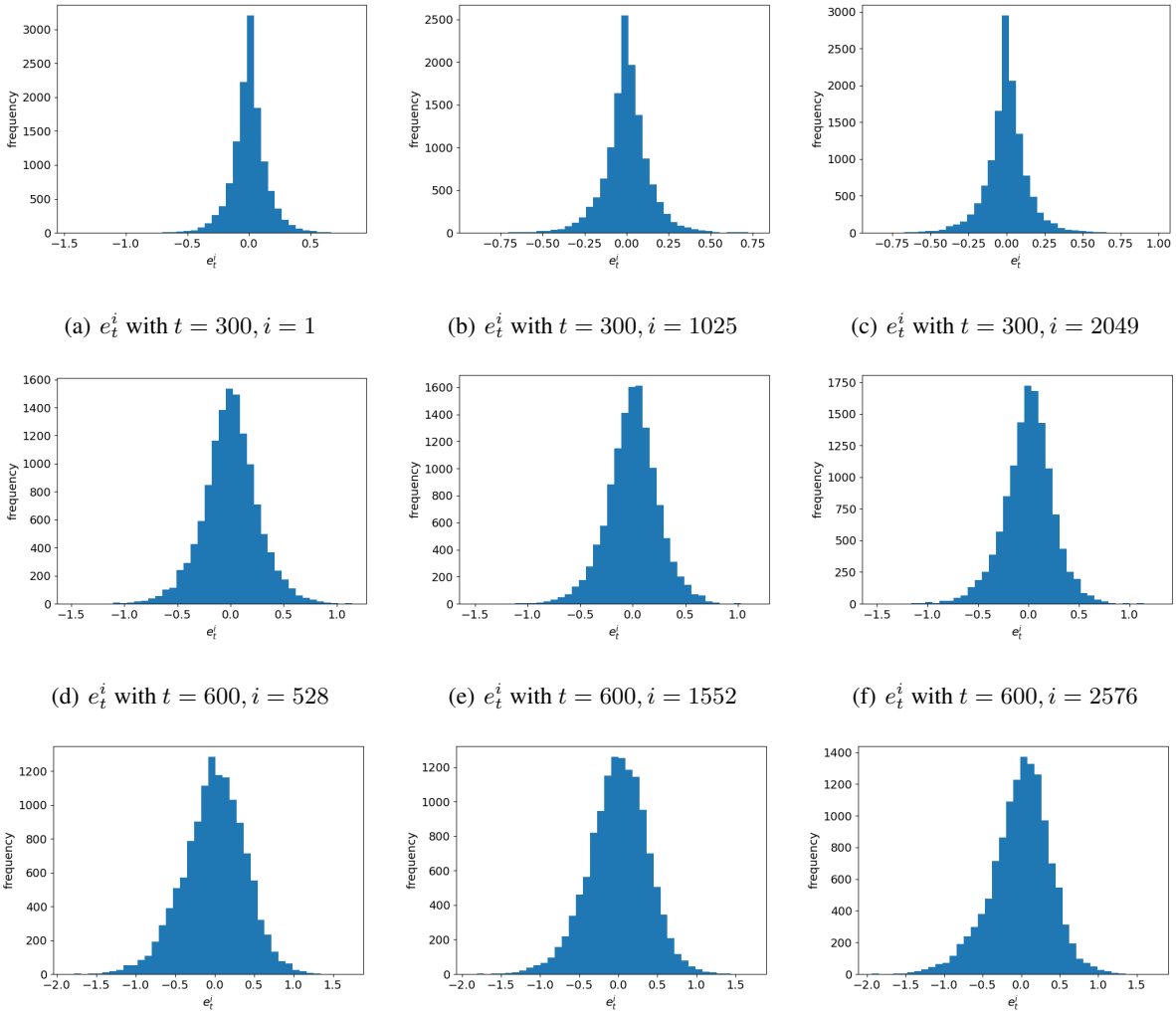
where the details of Tylor Series approximation are:

- Let $y = Wx + b$, we approximate $f(y)$ at the point $y = 0$.
- Hence, $f(y) \approx f(0) + f'(0)(y - 0) f(y) \approx f(0) + f'(0)(y - 0)$.
- Substitute y with y_1 and y_2, where $y_1 = Wx_1 + b, y_2 = Wx_2 + b$.
- Thus, $f(y_1) - f(y_2) \approx f(0) + f'(0)y_1 - f(0) - f'(0)y_2 = f'(0)(y_1 - y2) = f'(0)W(x_1 - x_2)$.

Since $\frac{\partial f}{\partial y}$ is bounded by 1 when $f = ReLU$, we can ignore it, and we have:

$$\|W(x_1 - x_2)\| \leq K \|x_1 - x_2\| . \tag{21}$$

We now introduce the Spectral Norm $\|W\|_2$. According to the definition $\|W\|_2 = \max_{x \neq 0} \frac{\|Wx\|}{\|x\|}$, we have:



(a) e_t^i with $t = 300, i = 1$ (b) e_t^i with $t = 300, i = 1025$ (c) e_t^i with $t = 300, i = 2049$

(d) e_t^i with $t = 600, i = 528$ (e) e_t^i with $t = 600, i = 1552$ (f) e_t^i with $t = 600, i = 2576$

(g) e_t^i with $t = 900, i = 1024$ (h) e_t^i with $t = 900, i = 2048$ (i) e_t^i with $t = 900, i = 3072$

Figure 6. The empirical distribution of e_t^i with different random values of t and i.

$$\|W(x_1 - x_2)\| \leq \|W\|_2 \cdot \|x_1 - x_2\| . \tag{22}$$

Comparing Eq. 21 with Eq. 22, we can use $\|W\|_2$ as the Lipschitz constant K. We can use the Frobenius Norm $\|W\|_F$ to approximate the Spectral Norm $\|W\|_2$ because, using the Cauchy inequality, we have:

$$\|Wx\| \leq \|W\|_F \cdot \|x\| , \tag{23}$$

where the definition of the Frobenius Norm is: $\|W\|_F = \sqrt{\sum_{i,j} w_{i,j}^2}$.

Thus, we can use the Frobenius Norm $\|W\|_F$ to approximate the constant K. Minimizing this constant during training is often implemented by adding a loss term $\lambda \|W\|_F^2$ to the loss function. This loss term is exactly the Weight Decay according to the definition of $\|W\|_F = \sqrt{\sum_{i,j} w_{i,j}^2}$.

A.7. Hyperparameters

For both ADM and ADM-IP, we use the hyperparameters specified in (Dhariwal & Nichol, 2021), except for LSUN tower, for which we used a resolution of 64×64. The hyperparameter values are reported in Tab. 9. We train all the models using the AdamW optimizer (Loshchilov & Hutter, 2019). Furthermore, we use 16-bit precision and loss-scaling (Micikevicius et al., 2017) for mixed precision training, but keeping 32-bit weights, EMA, and the optimizer state. We use an EMA rate of 0.9999 for all the experiments. These settings are the same as in (Dhariwal & Nichol, 2021).

We use Pytorch 1.8 (Paszke et al., 2019) and trained all the models on different NVIDIA Tesla V100s (16G memory). In more detail, we use 2 GPUs to train the models on CIFAR10 for 2 days, and 4 GPUs to train the models on ImageNet 32×32 for 34 days. For LSUN tower 64×64, CelebA 64×64 and FFHQ 128×128, we used 16 GPUs to train the models for 3 days, 5 days and 4 days, respectively.

Table 9. ADM and ADM-IP hyperparameter values

	CIFAR10 32×32	ImageNet 32×32	LSUN tower 64×64	CelebA 64×64	FFHQ 128×128
Diffusion steps	1,000	1,000	1,000	1,000	1,000
Noise schedule	cosine	cosine	cosine	cosine	cosine
Model size	57M	57M	295M	295M	543M
Channels	128	128	192	192	256
Residual blocks	3	3	3	3	3
Channels multiple	1, 2, 2, 2	1, 2, 2, 2	1, 2, 3, 4	1, 2, 3, 4	1, 1, 2, 3, 4
Heads channels	32	32	64	64	64
Attention resolution	16, 8	16, 8	32, 16, 8	32, 16, 8	32, 16, 8
BigGAN up/downsample	True	True	True	True	True
Dropout	0.3	0.3	0.1	0.1	0.1
Batch size	128	512	256	256	128
Training iterations	540K	5000K	340K	540K	480K
Training images	50K	1281K	708K	203K	70K
Learning rate	1e-4	1e-4	1e-4	1e-4	1e-4

Regarding the DDIM and DDIM-IP experiments, we use the default hyperparameters specified in the public code of Song et al. (2021a). We train both DDIM and DDIM-IP on CIFAR10 from scratch for 1600K iterations with batch size 128. The complete list of hyperparameters is shown in Tab. 10. We train DDPM/DDPM-IP with a single NVIDIA Tesla V100s (16G memory) for 8 days on a Pytorch 1.8 platform.

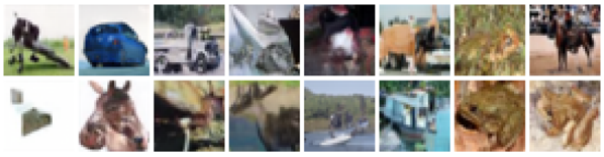
Table 10. DDIM and DDIM-IP hyperparameter values on CIFAR10 dataset

Diffusion Steps	1000	Variance type	fixed large
Noise schedule	linear	Ema rate	0.9999
Channels	128	Batch size	128
Channels multiple	1,2,2,2	Iterations	1600K
Residual blocks	2	Training images	50K
Attention resolution	16	Optimizer	Adam
Dropout	0.1	Learning rate	2e-4

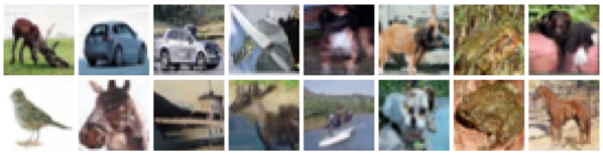
A.8. Qualitative Comparison between ADM and ADM-IP

In this section, we qualitatively compare ADM with ADM-IP. For a fair comparison, we start sampling the same $\boldsymbol{x}_T$ for both models. Fig. 7, 8, 9, 10, 11 show that the images generated by ADM-IP are usually comparable or better than those produced by ADM. For example, in Fig. 7, ADM fails to run into the bird, the boat and the dog modes in the first, the third and the sixth image on the second row. Similarly, in Fig. 9, ADM fails to complete the building in the fourth image on the second row. Moreover, the details and colors of the towers generated by ADM-IP are more visually realistic and appealing.

Finally, on the FFHQ 128×128 dataset, the ADM generated samples suffer from overexposure and loss of background detail, whereas the ADM-IP samples do not (see Fig. 11).



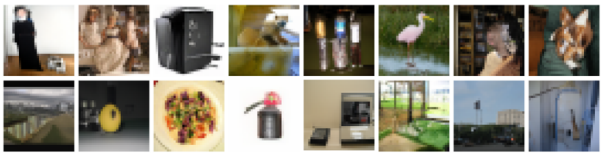
(a) Samples generated by ADM trained on CIFAR10 (FID 2.99)



(b) Samples generated by ADM-IP trained on CIFAR10 (FID 2.76)

Figure 7. CIFAR10, qualitative results. The samples are generated using 1,000 sampling steps.

(a) Samples generated by ADM trained on ImageNet 32×32 (FID 3.53)



(b) Samples generated by ADM-IP trained on ImageNet 32×32 (FID 2.72)

Figure 8. ImageNet 32×32, qualitative results. The samples are generated using 1,000 sampling steps.

A.9. Additional Qualitative Results for ADM-IP

We show additional images generated by our ADM-IP models trained on CIFAR10 (Fig. 12), ImageNet 32×32 (Fig. 13), LSUN tower 64×64 (Fig. 14), CelebA 64×64 (Fig. 15) and FFHQ 128×128 (Fig. 16). For each dataset, we used the best number of sampling steps as indicated in Tab. 3.

(a) Samples generated by ADM trained on LSUN tower 64×64 (FID 3.39)

(b) Samples generated by ADM-IP trained on LSUN tower 64×64 (FID 2.68)

Figure 9. LSUN tower 64×64, qualitative results. The samples are generated using 1,000 sampling steps.

(a) Samples generated by ADM trained on CelebA 64×64 (FID 1.60)

(b) Samples generated by ADM-IP trained on CelebA 64×64 (FID 1.31)

Figure 10. CelebA 64×64, qualitative results. The samples are generated using 1,000 sampling steps.

(a) Samples generated by ADM trained on FFHQ 128×128 (FID 9.65)

(b) Samples generated by ADM-IP trained on FFHQ 128×128 (FID 2.98)

Figure 11. FFHQ 128×128, qualitative results. The samples are generated using 1,000 sampling steps.

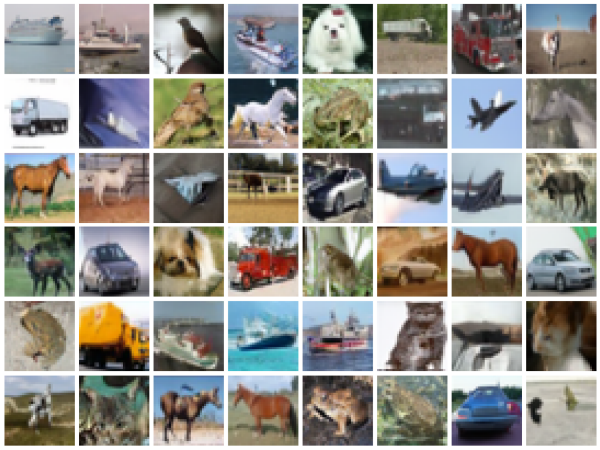


Figure 12. Samples generated by ADM-IP trained on CIFAR10 (FID 2.67 , 300 sampling steps)

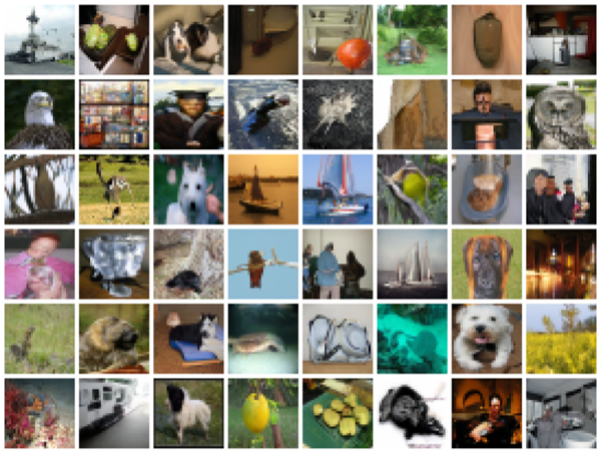


Figure 13. Samples generated by ADM-IP trained on ImageNet 32×32 (FID 2.66, 300 sampling steps)

Figure 14. Samples generated by ADM-IP trained on LSUN tower 64×64 (FID 2.60 , 300 sampling steps)

Figure 15. Samples generated by ADM-IP trained on CelebA 64×64 (FID 1.27, 900 sampling steps)

Figure 16. Samples generated by ADM-IP trained on FFHQ 128×128 (FID 2.98, 1,000 sampling steps)