INPUT PERTURBATION REDUCES EXPOSURE BIAS IN DIFFUSION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Denoising Diffusion Probabilistic Models (DDPMs) are fast becoming one of the dominant generative methods thanks to their high generation quality and diversity. However, one of the main problems of DDPMs is their large computational cost, which is due to the chain of sampling steps. In this paper, we argue that one of the reasons why DDPMs need a long sampling chain is due to an *exposure bias* problem, similar to the analogous 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. In order to alleviate this problem, we propose a very simple but effective training protocol modification, consisting in perturbing the ground truth samples to simulate the inference time prediction errors. We empirically show that the proposed input perturbation leads to a significant improvement of the sample quality and to smoother sampling chains, with a drastic acceleration of the inference time. For instance, in all the tested benchmarks, we observed an acceleration over a state-of-the-art DDPM of 12.5 times.

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 using a deep network, 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, can be used to train a deep denoising autoencoder ($\mu()$) 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 expensive computational time, which depends on the large number of steps T required at both the training and the inference stage. In this paper, we argue that one of the reasons why DDPMs need a large number of steps to learn to effectively sample, is the discrepancy between the training and the inference stages, 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, $\mu()$ 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, $\mu()$ 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, in the Image Captioning field, 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 order to alleviate this exposure bias problem, we propose a surprisingly simple but very effective method, which consists in explicitly modelling the prediction error during training. At training time, we perturb $\boldsymbol{x}_t$ and we feed $\mu()$ with a noisy version of $\boldsymbol{x}_t$, this way simulating the training-inference discrepancy, and forcing the autoencoder to learn 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). We empirically show that input perturbation largely improves the image generation quality, leading to smoother prediction trajectories at inference time. As a consequence, we can obtain the same generation quality of the baseline DDPMs either with a shorter training time and/or with a shorter sampling chain at inference stage. This is schematically illustrated in Fig. 1, where the red line represents a common full-chain sampling trajectory, while the green line represents a shorter trajectory in which random oscillations are reduced because the predictions change more gradually (i.e., close points in the input space correspond to similar output predictions).

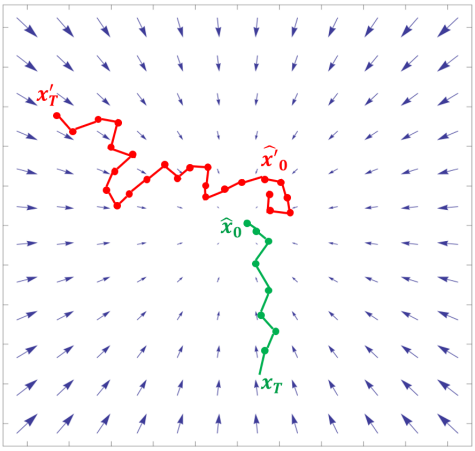


Figure 1: A schematic illustration of the benefit of the proposed input perturbation (green line) with respect to the inference sampling chain of a standard DDPM (red line).

Smoother prediction trajectories can also be obtained by explicitly encouraging the function $\mu()$ to be Lipschitz continuous (Sec. 5.1). The rationale behind this is that a Lipschitz continuous function $\mu()$ generates small prediction differences between neighborhood points of its domain, and this leads to a DDPM which is more robust to the inference-time errors. However, the explicit minimization of the Lipschitz constant in DDPMs is either very time consuming or less effective than input perturbation (Sec. 5.1, 7). In contrast, directly perturbing the prediction network input at training time has no additional training overhead and can be obtained without using additional regularization terms. We denote our method as *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) and speed up the inference-time sampling. On the Cifar10 (Krizhevsky et al., 2009), the ImageNet 32×32 (Chrabaszcz et al., 2017) and the LSUN 64×64 (Yu et al., 2015) datasets, DDPM-IP, with only 80 sampling steps, generates lower FID scores that the state-of-the-art ADM Dhariwal & Nichol (2021) with 1,000 steps, which corresponds to an acceleration of more than 12.5 times. Finally, DDPM-IP can be implemented with just two lines of code, without any change in the network architecture or the loss function, and thus it can be easily plugged in different existing DDPMs.

In summary, our contributions are:

- We argue that there is an exposure bias problem in DDPMs which has not been investigated so far. To alleviate this problem, we propose a very simple method based on the input perturbation at training time.
- We additionally propose other solutions to the exposure bias problem which are based on Lipschitz continuous prediction networks. However, DDPM-IP leads to a higher sample quality and a faster sampling.

- Using an image domain and common experimental benchmarks, we show that our method can significantly improve the generation quality and drastically speed up the inference time sampling.

2 RELATED WORK

DDPMs have been introduced by Sohl-Dickstein et al. (2015) and later improved in (Nichol & Dhariwal, 2021; Ho et al., 2020). More recently, Dhariwal & Nichol (2021) have shown that DDPMs can achieve still image generation results on par or superior to the best Generative Adversarial Network (GAN) approaches (Goodfellow et al., 2014). Similarly to GANs, the generation process in DDPMs can be both unconditional and conditioned. For instance, GLIDE (Nichol et al., 2022) is one of the state-of-the-art text-guided image generation methods, in which a DDPM learns to generate images according to an input textual sentence. Differently from GLIDE, where the diffusion model is defined on the input space (i.e., 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-driven image manipulation is explored also by Kim & Ye (2021), who use the CLIP space as a guidance for the backward diffusion process (more details on the latter in Sec. 3). 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 architecture or loss function, 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 design choices.

The specific problem of increasing the convergence speed and/or reducing the number of sampling steps T (Sec. 1) has been thoroughly investigated due to its practical implications. For instance, Song et al. (2021) propose a non-Markovian diffusion process in which the prediction network directly predicts the initial point of the sequence ($\hat{\boldsymbol{x}}_0$) and then $\hat{\boldsymbol{x}}_0$ is used to obtain a new sample of the chain $\hat{\boldsymbol{x}}_{t-1}$, and they empirically show that this leads to a reduced number of sampling steps at inference time. Salimans & Ho (2022) propose to distill the prediction network into new networks which progressively reduce the number of required inference sampling steps. However, the disadvantage of this approach is the need of training multiple networks. Rombach et al. (2021) propose to speed up the DDPM sampling by splitting the process in a compression stage and a generation stage, and applying the DDPM on the compressed (latent) space rather than directly on the pixel space. Hoogeboom et al. (2022) propose 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 and which leads to a partially parallelized sampling at inference time. Recently, Chen et al. (2021) found that, instead of conditioning the prediction network ($\mu()$) on a discrete diffusion step t, it is beneficial to condition $\mu()$ on a continuous noise level. Our approach is orthogonal to these previous works, and it can potentially be used jointly with any of them.

3 BACKGROUND

Without loss of generality, we assume an image domain and we focus on DDPMs which directly define a diffusion process on the input space (see Sec. 2 for DDPM examples defined on a latent space or based on different domains). Moreover, following (Nichol & Dhariwal, 2021; Dhariwal & Nichol, 2021), we assume that the pixel values are linearly scaled to $[-1, 1]$. Given samples from a real data distribution $q(\boldsymbol{x}_0)$, a DDPM defines a *forward process* and a *reverse process*. The forward process, defined as a Markov chain, starts from a real image $\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$ and iteratively adds Gaussian noise for T diffusion steps:

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

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

where $\beta_1, ..., \beta_T$ is a noise schedule and Eq. 1-2 progressively destroy the original image $\boldsymbol{x}_0$ until obtaining a completely noisy image $\boldsymbol{x}_T$. On the other hand, the reverse process is based on a prediction network which learns to invert the forward process. More formally, starting from random noise $\boldsymbol{x}_T \sim p(\boldsymbol{x}_T) = \mathcal{N}(\mathbf{0}, I)$, the reverse process is defined by means of learned transition probabilities parameterized by $\boldsymbol{\theta}$ and given by:

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

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T}) = p(\boldsymbol{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t), \tag{4}$$

where $\sigma_1, ...\sigma_T$ is a predefined noise schedule and $\sigma_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$, being $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$ and $\alpha_i = 1 - \beta_i$. Given $\boldsymbol{x}_0$, $\boldsymbol{x}_t$ can be obtained (Ho et al., 2020) by:

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}, \tag{5}$$

where $\boldsymbol{\epsilon}$ is a noise vector ($\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I)$). Instead of using a prediction network which predicts 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 $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}()$ which predicts the noise vector ($\boldsymbol{\epsilon}$). Using $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}()$, and a simple $L2$ loss function, the training objective becomes:

$$L(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0), \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I), t \sim \mathbb{U}(\{1, ..., T\})}[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)||^2]. \tag{6}$$

Note that, in Eq. 6, $\boldsymbol{x}_t$ and $\boldsymbol{\epsilon}$ are ground-truth terms, while $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ is the network prediction. Using Eq. 6, the training and the sampling algorithms are described in Alg. 1-2, respectively.

Algorithm 1 Training

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

Algorithm 2 Sampling

1: $\hat{\boldsymbol{x}}_T \sim \mathcal{N}(\mathbf{0}, I)$
2: **for** $t := T, ..., 1$ **do**
3: $\quad$ if $t > 1$ then $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, I)$, else $\boldsymbol{z} = \mathbf{0}$
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 THE EXPOSURE BIAS PROBLEM IN DIFFUSION MODELS

Comparing line 6 of Alg. 1 with line 4 of Alg. 2, we observe that the inputs of the prediction network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}()$ are different between the training and the inference phase. Specifically, at training time, standard DDPMs use $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$, where $\boldsymbol{x}_t$ is a *ground truth* sample (Eq. 5). 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 $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}()$ at the previous sampling step $(t+1)$. As mentioned in Sec. 1, this leads to a training-inference discrepancy, which is similar to, e.g., the exposure bias problem observed 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 this discrepancy, we propose a simple experiment in which we measure the difference between the ground truth $\boldsymbol{x}_0$ and the predicted $\hat{\boldsymbol{x}}_0$ in the pixel space. The complete algorithm is described in Alg. 3. Given a real image $\boldsymbol{x}_0$, we first compute $\boldsymbol{x}_t$ by Eq. 5, then we use the reverse process of a *pre-trained* network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$. Specifically, $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ is trained using the standard algorithm (Alg. 1) and then frozen. To reduce the stochasticity of the sampling chain, we adopt the equation in line 4 of Alg. 2 but removing the stochastic term $\sigma_t \boldsymbol{z}$. This deterministic reverse diffusion

process decreases the generation diversity and it enables the model to target at the mode of $\boldsymbol{x}_0$ rather than falling into other modes (images) Luo (2022). Finally, we use the $L1$-distance between $\boldsymbol{x}_0$ and $\hat{\boldsymbol{x}}_0$ in the pixel space to estimate the cumulative error computed in the whole trajectory of t steps. In this experiment, we use ADM (Dhariwal & Nichol, 2021) and the ImageNet 32×32 dataset (Chrabaszcz et al., 2017).

Algorithm 3 Measuring the inference-time prediction error

Initialize $\delta_t = 0$, $n_t = 0$ $(\forall t \in \{1, ..., T\})$
repeat
  $\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$
  $t \sim \mathbb{U}(\{1, ..., T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I)$
  compute $\boldsymbol{x}_t$ using Eq. 5
  for $\tau := t, ..., 1$ **do**
    $\hat{\boldsymbol{x}}_{\tau-1} = \frac{1}{\sqrt{\alpha_\tau}}(\hat{\boldsymbol{x}}_\tau - \frac{1-\alpha_\tau}{\sqrt{1-\bar{\alpha}_\tau}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}}_\tau, \tau))$
  end for
  $\delta_t := \delta_t + ||\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0||_1$
  $n_t := n_t + 1$
until N iterations
$\forall t \in \{1, ..., T\}, n_t \neq 0 : \bar{\delta}_t = \frac{\delta_t}{n_t}$

In Tab. 1, we show the training-inference discrepancy measured using $\bar{\delta}_t$ with respect to different trajectory lengths (t). This table shows that the discrepancy grows up with the increase of the number of reverse diffusion steps. In other words, the errors most likely *accumulate* over different steps during sampling. In the Appendix A.2, we visualize a few sample images $\boldsymbol{x}_0$ used in this experiment, jointly with the corresponding predicted images $\hat{\boldsymbol{x}}_0$, which further illustrates the strong increase of the error with respect to the number of reverse time steps t.

Table 1: A measure of the error accumulation ($\bar{\delta}_t$) with respect to different prediction trajectory lengths. Note that, due to the pixel scaling, the error is upper bounded by 2.

Model	Number of reverse diffusion steps			
	100	300	600	1000
ADM (baseline)	0.0539	0.1074	0.1812	0.8165

5 Using input perturbation to alleviate the training-inference discrepancy

The solution we propose to alleviate the training-inference discrepancy (Sec. 4) is very simple: at training time we explicitly model the prediction error using input perturbation. 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 vector $\boldsymbol{x}_t$ (computed using Eq. 5). Hence, given a random noise vector $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, I)$, we create a perturbed version ($\boldsymbol{y}_t$) of the input ground truth:

$$\boldsymbol{y}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t}(\boldsymbol{\epsilon} + \gamma_t\boldsymbol{\xi}). \tag{7}$$

For simplicity, we use a flat noise schedule for $\boldsymbol{\xi}$ and we set $\gamma_0 = ..., \gamma_T = \gamma$. 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). Hence, to avoid adding a second noise schedule $(\gamma_0, ..., \gamma_T)$ to the training procedure, we opted for a simpler, although most likely sub-optimal, solution, in which γ_t does not vary depending on t. In Alg. 4 we show the proposed training algorithm, in which $\boldsymbol{x}_t$ is replaced by $\boldsymbol{y}_t$.

Note that, in line 8, we use $\boldsymbol{y}_t$ as input of the prediction network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ but we keep using $\boldsymbol{\epsilon}$ as the regression target. In other words, the new additive noise term ($\boldsymbol{\xi}$) we introduce is used *asymmetrically*, because it is *not* included in the prediction target ($\boldsymbol{\epsilon}$). For this reason, the proposed training protocol (Alg. 4) 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. 5) and as the target of the prediction network (line 6 of Alg. 1). Finally, at inference time, we use Alg. 2 without any change.

Algorithm 4 DDPM-IP: Training with input perturbation

1: **repeat**
2: $\quad \boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$
3: $\quad t \sim \mathbb{U}(\{1, ..., T\})$
4: $\quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I)$
5: $\quad \boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, I)$
6: $\quad$ compute $\boldsymbol{y}_t$ using Eq. 7
7: $\quad$ take a gradient descent step on
8: $\quad\quad \nabla_{\boldsymbol{\theta}} ||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}_t, t)||^2$
9: **until** converged

5.1 LIPSCHITZ CONTINUOUS FUNCTIONS

The proposed input perturbation resembles Denoising Autoencoders Vincent et al. (2008) which learn a smooth prediction function with small gradients around each data sample Goodfellow et al. (2016). Specifically, in the procedure described in Sec. 5, we train the network to be more robust the prediction errors by randomly perturbing its input ($\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}_t, \cdot)$) while simultaneously asking to predict the original noise vector $\boldsymbol{\epsilon}$ independently of the added noise ($\gamma\sqrt{1-\bar{\alpha}_t}\boldsymbol{\xi}$). Since the perturbation is *locally* distributed around the ground truth $\boldsymbol{x}_t$ ($\boldsymbol{y}_t \sim \mathcal{N}(\boldsymbol{x}_t, \gamma^2(1-\bar{\alpha}_t)I)$), the simplest way to solve this task for the network is to *smooth* the prediction output with respect to its input.

An alternative way to obtain smooth predictions which mitigate the effects of the inference-time prediction errors is to explicitly encourage $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}()$ to be Lipschitz continuous, i.e. to satisfy:

$$||\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}, t) - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{y}, t)|| \leq K||\boldsymbol{x} - \boldsymbol{y}||, \quad \forall(\boldsymbol{x}, \boldsymbol{y}) \tag{8}$$

for a small constant K. We implemented this idea using two standard Lipschitz constant minimization methods, i.e., *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 prediction network input and we use the original training algorithm (Alg. 1), with the only difference being the loss function used in line 6, where the $L2$ loss is used jointly with a regularization term, as 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:

$$L_{GP}(\boldsymbol{\theta}) = ||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)||^2 + \lambda_{GP} \left\| \frac{\partial \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)}{\partial \boldsymbol{x}} \right\|_F^2, \tag{9}$$

where λ_{GP} is the weight of the gradient penalty term, and the latter is used to encourage $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}()$ to have a low Lipschitz constant Rifai et al. (2011). However, a gradient penalty regularization is very slow Yoshida & Miyato (2017) because it involves one forward and two backward passes for each training image: We first need to compute $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ and backpropagate the corresponding gradients through the network till the input layer, then we can compute the whole loss function (Eq. 9) and finally we need to backpropagate the loss gradients ($\nabla_{\boldsymbol{\theta}} L_{GP}(\boldsymbol{\theta})$) to update the weights ($\boldsymbol{\theta}$). In our preliminary experiments using this approach, we observed that, despite having encouraging results in terms of the quality of the generated images, the training time increased about three times with respect to the baseline. This is also due to the Pytorch implementation, in which the computational graph should keep in memory much more information than in the case in which there is only one forward and one backward pass. Since our goal is to speed up the training (besides the testing) time of DDPMs, this approach is clearly counterproductive.

Weight decay. As shown in (Liu et al., 2022), Lipschitz continuity can also be encouraged using a weight decay regularization (see Appendix A.1 for more details). In this case, the final loss is:

$$L_{WD}(\boldsymbol{\theta}) = ||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)||^2 + \lambda_{WD}||\boldsymbol{\theta}||^2, \tag{10}$$

where λ_{WD} is the weight of the regularization term. Weight decay is simple to implement and computationally efficient. However, it involves changing the loss function, which may be not easy to do in some domain-specific DDPMs. Moreover, in Sec. 7 we compare this approach with the input perturbation presented in Sec. 5 and we show that, using the latter, the generation quality improves.

6 MAIN RESULTS

To evaluate the effectiveness of DDPM-IP, we use the state-of-the-art diffusion model ADM Dhariwal & Nichol (2021) (without classifier guidance) as the baseline. The generation quality is estimated using standard metrics: the Fréchet Inception Distance (FID) (Heusel et al., 2017) and 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 coherent high-level structure. In our experiments, we focus on image generation tasks and train the models on the Cifar10 Krizhevsky et al. (2009), the ImageNet 32×32 Chrabaszcz et al. (2017) and the LSUN tower 64×64 Yu et al. (2015) dataset.

Following prior work (Ho et al., 2020; Nichol & Dhariwal, 2021), we produce 50K samples for each trained model and we use the full training set to compute the reference distribution statistics, except for LSUN tower where we use 50K training samples as the reference data and produce 10K samples. Regarding the main hyperparameters, we choose $T = 1000$ for training in all the experiments, while we use the *time step respacing* technique (Nichol & Dhariwal, 2021) to generate samples with different numbers of times teps (see Tab. 2). Moreover, we empirically found that $\gamma = 0.1$ (Sec. 5) is optimal in both Cifar10 and ImageNet 32×32, and we used $\gamma = 0.1$ in all the experiments and the datasets (including LSUN, in which the value of γ was not tuned). The complete list of hyperparameters (e.g. the learning rate, the batch size, etc.) and network architecture setting, which are shared by all the tested models, can be found in Appendix A.3.

The results of all the models are shown in Tab. 2. This table shows that, independently of the dataset, the metric (either FID or sFID), and the number of sampling steps (T), DDPM-IP is *always* significantly better that ADM, sometimes drastically better. For instance, on LSUN with $T = 80$, we have a more than 5 sFID score improvement with respect to ADM, tested with the same number of sampling steps. Moreover, DDPM-IP converges much faster than the baseline model during training in the LSUN tower dataset, where DDPM-IP converges at 200K training iterations while ADM saturates around 300K iterations (see Fig. 2). A similar phenomenon is also observed while training of Cifar10. On ImageNet 32×32, although the two models converge at similar iterations, DDPM-IP consistently achieves a lower FID along the whole training. Furthermore, on this dataset, DDPM-IP reaches the same FID score (with $T = 1000$) as the final ADM FID scores in much less training time. For example, our model reaches FID 3.83 at 1000K training iterations whereas the baseline model gets FID 3.78 at 2500K iterations, showing a 2.5x speed-up in the training time.

Tab. 2 also shows that DDPM-IP can drastically accelerate the inference time, obtaining better results than the baseline with shorter sampling trajectories. For example, using only 80 sampling time steps, in all three datasets, both the FID and the sFID scores of DDPM-IP are better than the corresponding baseline values (independently of the number of sampling steps used for the latter). Moreover, comparing the DDPM-IP results with 80 sampling steps with the results of ADM with a standard number of sampling steps (i.e., $T = 1000$), DDPM-IP is almost always better (except the comparable values of sFID on ImageNet 32×32), which leads to a remarkable 12.5x speed-up of the inference stage.

Another interesting finding is that the best FID and sFID scores of both ADM and DDPM-IP are obtained in the range from 100 to 300 sampling steps, while the models are all trained with 1000 diffusion steps. A similar phenomenon was also observed by Nichol & Dhariwal (2021), enve though they did not provide an explanation. We believe that exposure bias problem may contribute to explain this apparently counterintuitive phenomenon in which fewer sampling steps lead to a better generation quality. Indeed, while on the one hand more sampling steps correspond to a diffusion

Table 2: The best FID and sFID using different sampling time steps

Sampling time steps	Model	Cifar10 32×32		ImageNet 32×32		LSUN tower 64×64	
		FID	sFID	FID	sFID	FID	sFID
1000	ADM (baseline)	3.58	4.05	3.6	3.3	4.66	17.21
	DDPM-IP	**3.25**	**3.75**	**2.87**	**2.39**	**4.05**	**15.63**
300	ADM	3.47	4.12	3.58	3.48	4.54	17.46
	DDPM-IP	**3.14**	**3.72**	**2.74**	**2.58**	**3.79**	**15.56**
100	ADM	3.56	4.42	4.26	4.48	4.79	19.51
	DDPM-IP	**3.12**	**3.86**	**3.24**	**3.13**	**4.13**	**16.11**
80	ADM	3.74	4.66	4.61	4.76	5.66	21.58
	DDPM-IP	**3.26**	**3.89**	**3.57**	**3.33**	**4.27**	**16.51**

process which can be more easily approximated with Gaussian distributions (Ho et al., 2020), on the other hand, longer sampling trajectories correspond to a larger accumulation of the prediction errors. Hence, the range $[100, 300]$ leads to the best generation quality because, presumably, it is a good trade-off between these two opposite aspects.

7 Comparison with explicit Lipschitz continuous functions

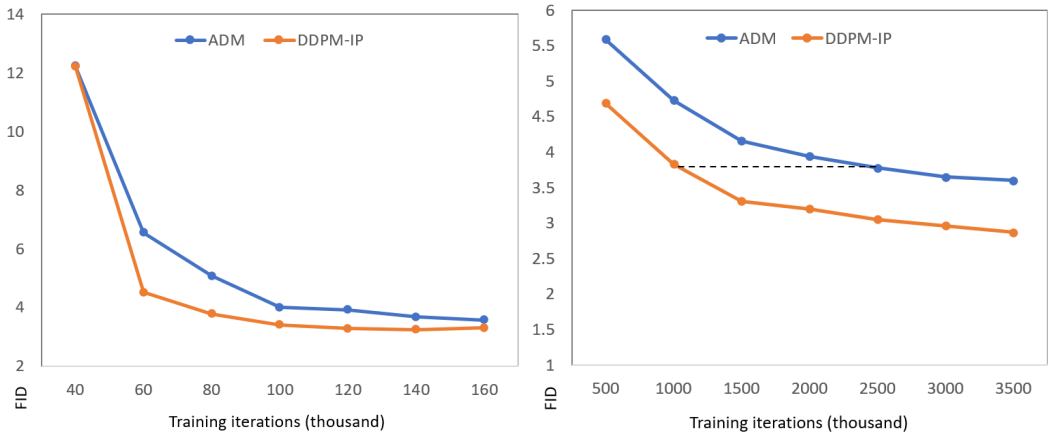
In this section, we compare DDPM-IP with the two alternative methods proposed in Sec. 5.1 which are based on an explicit Lipschitz constant minimization. We use "ADM-WD" to indicate the weight decay method and "ADM-GP" for the gradient penalty approach. The corresponding loss weights we used are: $\lambda_{GP} = 1e-6$ and $\lambda_{WD} = 0.03$. For this experiment, we use Cifar10 because ADM-GP is too time consuming to be trained on larger datasets (Sec. 5.1). The results in Tab. 3 show that all the three models outperform the baseline, but DDPM-IP gets the best FID and sFID scores. These results show the effectiveness of Lipschitz continuous functions in boosting the image quality generated by DDPMs.

Table 3: FID and sFID scores using 1000 sampling steps.

Model	Cifar10 32×32	
	FID	sFID
ADM (baseline)	3.58	4.05
ADM-GP	3.28	3.92
ADM-WD	3.34	3.94
DDPM-IP	**3.25**	**3.75**

8 Conclusions

In this paper, we proposed DDPM-IP, a training protocol for DDPMs which is based on input perturbation to explicitly model the prediction errors and alleviate the DDPM exposure bias problem. We empirically showed that DDPM-IP can significantly improve the image quality and *drastically* reduce the number of sampling steps at 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.



(a) Cifar10

(b) ImageNet 32×32

(c) LSUN 64×64

Figure 2: The FID scores computed using models trained with different numbers of training iterations. At inference time, for all the models and all the training iterations, we use $T = 1000$ sampling steps. The dashed line in (b) indicates that DDPM-IP, trained with 1000K iterations, reaches the same FID scores as ADM when trained with almost 2500K iterations.

REFERENCES

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *NeurIPS*, 2015.

Nanxin Chen, Yu Zhang, Heiga Zen, Ron J. Weiss, Mohammad Norouzi, and William Chan. Wavegrad: Estimating gradients for waveform generation. In *ICLR*, 2021.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv:1707.08819*, 2017.

Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah. Diffusion models in vision: A survey. *arXiv preprint arXiv:2209.04747*, 2022.

Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat GANs on image synthesis. In *NeurIPS*, 2021.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Shuyang Gu, Dong Chen, Jianmin Bao, Fang Wen, Bo Zhang, Dongdong Chen, Lu Yuan, and Baining Guo. Vector quantized diffusion model for text-to-image synthesis. *arXiv:2111.14822*, 2021.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *NeurIPS*, 30, 2017.

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. *NeurIPS*, 30, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. In *NeurIPS*, 2021.

Emiel Hoogeboom, Alexey A. Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. In *ICLR*, 2022.

Gwanghyun Kim and Jong Chul Ye. DiffusionCLIP: text-guided image manipulation using diffusion models. *arXiv:2110.02711*, 2021.

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

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

Hsueh-Ti Derek Liu, Francis Williams, Alec Jacobson, Sanja Fidler, and Or Litany. Learning smooth neural functions via lipschitz regularization. *arXiv preprint arXiv:2202.08345*, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *ICLR*, 2019.

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

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv:1710.03740*, 2017.

Gautam Mittal, Jesse H. Engel, Curtis Hawthorne, and Ian Simon. Symbolic music generation with diffusion models. In *Proceedings of the 22nd International Society for Music Information Retrieval Conference, ISMIR*, 2021.

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:2103.03841*, 2021.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *ICML*, 2021.

Alexander Quinn Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. GLIDE: towards photorealistic image generation and editing with text-guided diffusion models. In *ICML*, 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 32, 2019.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *ICML*, 2021.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents. *arXiv:2204.06125*, 2022.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *ICLR*, 2016.

Kashif Rasul, Calvin Seward, Ingmar Schuster, and Roland Vollgraf. In Marina Meila and Tong Zhang (eds.), *ICML*, 2021.

Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *CVPR*, 2017.

Salah Rifai, Xavier Muller Pascal Vincent, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML*, 2011.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *arXiv:2112.10752*, 2021.

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

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

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, 2015.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2021.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, pp. 1096–1103, 2008.

Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics. In *ICML*, 2011.

Ling Yang, Zhilong Zhang, and Shenda Hong. Diffusion models: A comprehensive survey of methods and applications. *arXiv:2209.00796*, 2022.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, 2019.

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

Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

A Appendix

A.1 Relation between the Lipschitz constant minimization and weight decay minimization

For functions that satisfy a low Lipschitz constant K, the output difference $\|f_w(x_1) - f_w(x_2)\|$ should be small if the input difference $\|x_1 - x_2\|$ is small, which is expressed as follows:

$$\|f_w(x_1) - f_w(x_2)\| \leq K \cdot \|x_1 - x_2\| \tag{11}$$

Since a neural network is usually a stack of layers, we consider a single layer neural network, $f(x) = ReLU(Wx + b)$, thus we have:

$$\|f(Wx_1 + b) - f(Wx_2 + b)\| \leq K \cdot \|x_1 - x_2\| \tag{12}$$

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

$$\left\| \frac{\partial f}{\partial y} \cdot W(x_1 - x_2) \right\| \leq K \cdot \|x_1 - x_2\|, \tag{13}$$

where the details of Tylor Series approximation are:

- let $y = Wx + b$, we approximate $f(y)$ at the point $y = 0$
- therefore, $f(y) \approx f(0) + f'(0)(y - 0)$
- 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:

$$\|W(x_1 - x_2)\| \leq K \|x_1 - x_2\|. \tag{14}$$

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

$$\|W(x_1 - x_2)\| \leq \|W\|_2 \cdot \|x_1 - x_2\| \tag{15}$$

Comparing Eq.14 with Eq.15, we can use $\|W\|_2$ as the Lipschitz constant K. For the simple case, we can use the Frobenius Norm $\|W\|_F$ to approximate the Spectral Norm $\|W\|_2$ because, using the Cauchy inequality, we have:

$$\|Wx\| \leq \|W\|_F \cdot \|x\|, \tag{16}$$

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.2 Exposure bias

We visualize the ground truth $\boldsymbol{x}_0$ and the predicted $\hat{\boldsymbol{x}}_0$ under different diffusion steps. Fig. 3 shows that the distance between $\boldsymbol{x}_0$ and $\hat{\boldsymbol{x}}_0$ increases with the diffusion chain getting longer.

A.3 Hyperparameters

Since ADM Dhariwal & Nichol (2021) is the baseline model in this paper, we keep the hyperparameters consistent with the parameters in ADM, except that we only train LSUN in the resolution of 64×64. The hypterparameters for training the diffusion models are shown in Tab. 4. We train all of our models using AdamW Loshchilov & Hutter (2019) optimizer. Furthermore, 16-bit precision and loss-scaling Micikevicius et al. (2017) are adopted for mixed precision training, but maintain

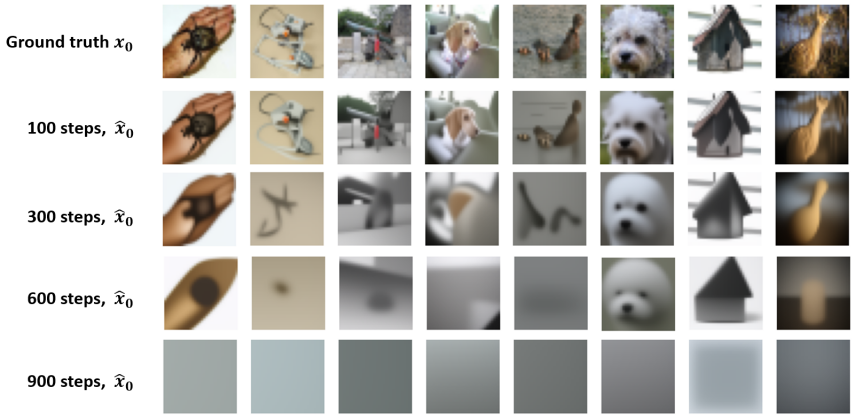


Figure 3: Visualization of the exposure bias problem at different diffusion chain lengths.

32-bit weights, EMA, and optimizer state. We use an EMA rate of 0.9999 for all experiments. These settings are the same as the configuration in Dhariwal & Nichol (2021).

As for the platform and training cost, we use Pytorch 1.8 Paszke et al. (2019) and train all the models on several NVIDIA Tesla V100s (16G memory). In details, we use 2 GPUs to train the models on Cifar10 for 1 day, use 4 GPUs to train the models on ImageNet 32×32 for 23 days, and use 16 GPUs to train the models on LSUN tower 64×64 for 3 days.

Table 4: Hyperparameters for diffusion models

	Cifar10 32×32	ImageNet 32×32	LSUN tower 64×64
Diffusion steps	1000	1000	1000
Noise schedule	cosine	cosine	cosine
Model size	57M	57M	295M
Channels	128	128	192
# Residual blocks	3	3	3
Channels multiple	1,2,2,2	1,2,2,2	1,2,3,4
Heads channels	32	32	64
Attention resolution	16, 8	16, 8	32, 16, 8
BigGAN up/downsample	True	True	True
Dropout	0.3	0.3	0.1
Batch size	128	512	256
Iterations	200K	3500K	340K
# Training images	60K	1281K	708K
Learning rate	1e-4	1e-4	1e-4

A.4 SAMPLES

We show some random image samples generated by our DDPM-IP models below. The sampling steps are fixed at 300 since it produces the best image quality among Cifar10, ImageNet 32×32 and LSUN tower 64×64 datasets.

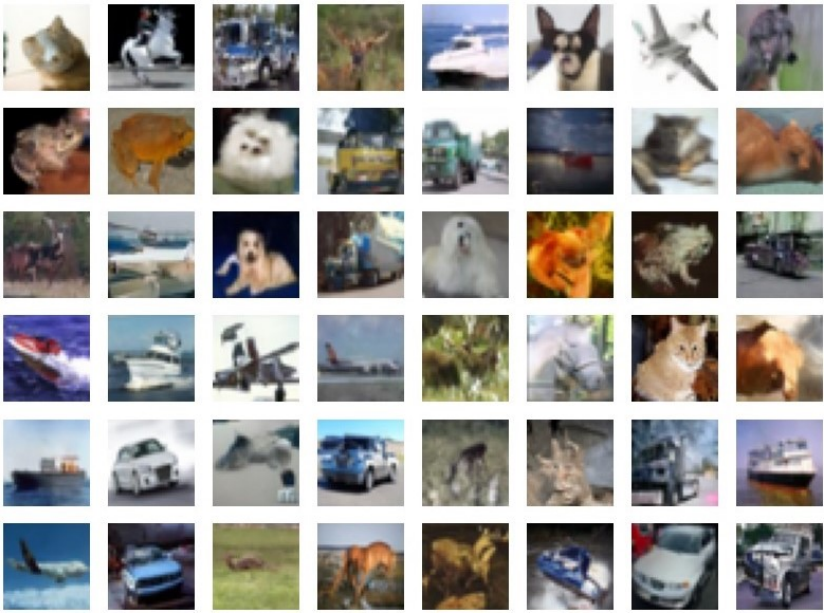


Figure 4: A few samples generated by DDPM-IP using Cifar10 (FID 3.14, 300 sampling steps)

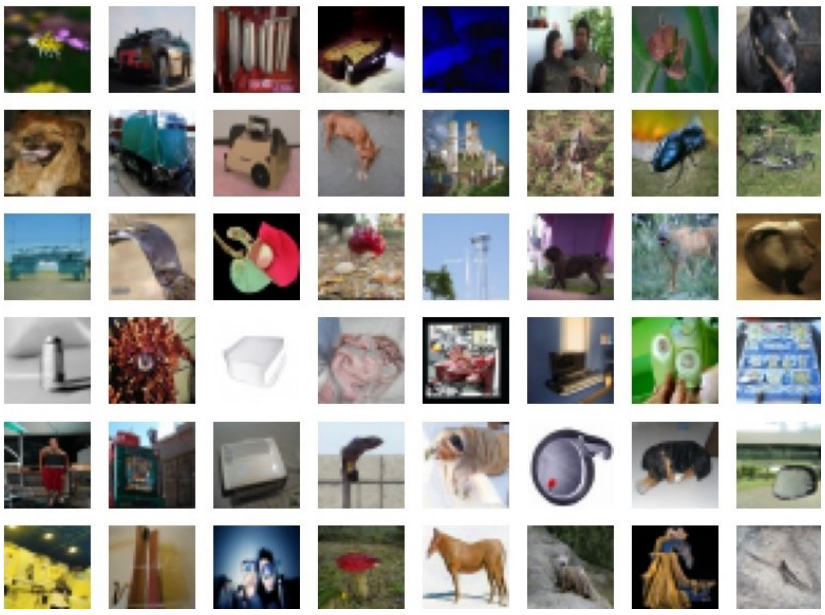


Figure 5: A few samples generated by DDPM-IP using ImageNet 32×32 (FID 2.74, 300 sampling steps)

Figure 6: A few samples generated by DDPM-IP using LSUN tower 64×64 (FID 3.79 , 300 sampling steps)