A Stochastic differential equation (SDE)

In (Song et al., 2021c), three types of SDE diffusion processes are presented. Depending on the type, $f(\mathbf{x}, t)$ and $g(t)$ are defined as follows:

$$f(\mathbf{x}, t) = \begin{cases} 0, & \text{if VE-SDE,} \\ -\frac{1}{2}\beta(t)\mathbf{x}, & \text{if VP-SDE,} \\ -\frac{1}{2}\beta(t)\mathbf{x}, & \text{if sub-VP-SDE,} \end{cases} \tag{8}$$

$$g(t) = \begin{cases} \sqrt{\frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}}, & \text{if VE-SDE,} \\ \sqrt{\beta(t)}, & \text{if VP-SDE,} \\ \sqrt{\beta(t)(1 - e^{-2\int_0^t \beta(s)\,\mathrm{d}s})}, & \text{if sub-VP-SDE,} \end{cases} \tag{9}$$

where $\sigma^2(t)$ and $\beta(t)$ are functions w.r.t. time t. Full derivatives of VE, VP and sub-VP SDE are presented in Song et al. (2021c, Appendix. B).

B Experimental details

In this section, we describe the detailed experimental environments of SPI-GAN. We build our experiments on top of Kang et al. (2022)

B.1 Experimental environments

Our software and hardware environments are as follows: Ubuntu 18.04 LTS, Python 3.9.7, PyTorch 1.10.0, CUDA 11.1, NVIDIA Driver 417.22, i9 CPU, and NVIDIA RTX A6000.

B.2 Target diffusion model

Our model uses a forward SDE to transform an image ($\mathbf{x}_0$) into a noise vector ($\mathbf{x}_T$). When generating a noise vector, we use the forward equation of VP-SDE for its high efficacy/effectiveness. The $\beta(t)$ function of VP-SDE is as follows:

$$\beta(t) = \beta_{\text{min}} + t(\beta_{\text{max}} - \beta_{\text{min}}) \tag{10}$$

where β_{max} = 20, β_{min} = 0.1, and $t' := \frac{t}{T}$ which is normalized from $t \in \{0, 1, \dots, T\}$ to $[0, 1]$. Under these conditions, Song et al. (2021c, Appendix B) proves that the noise vector at $t' = 1$ ($\mathbf{x}_T$) follows a unit Gaussian distribution.

B.3 Data augmentation

Our model uses the adaptive discriminator augmentation (ADA) (Karras et al., 2020a), which has shown good performance in StyleGAN2.[1] The ADA applies image augmentation adaptively to training the discriminator. We can determine the maximum degree of the data augmentation, which is known as an ADA target, and the number of the ADA learning can be determined through the ADA interval. We also apply mixing regularization (λ_{mixing}) to encourage the styles to localize. Mixing regularization determines how many percent of the generated images are generated from two noisy images during training (a.k.a, style mixing). There are hyperparameters for the data augmentation in Table 9.

B.4 Model architecture

Our proposed model is similar to StyleGAN2. However, StyleGAN2 architecture is modified to implement our proposed straight-path interpolation after adding the NODE-based mapping network and customizing some parts.

Mapping network. Our mapping network consists of two parts. First, the network architecture to define the function $\mathbf{o}$ is in Table 7. Second, the NODE-based network has the following ODE function $\mathbf{r}$ in Table 8.

[1] `https://github.com/NVlabs/stylegan2` (Nvidia Source Code License)

Table 7: The architecture of the network $\mathbf{o}$.

Layer	Design	Input Size	Output Size
1	LeakyReLU(Linear)	$C \times H \times W$	$\dim(\mathbf{h})$

Table 8: The architecture of the network $\mathbf{r}$.

Layer	Design	Input Size	Output Size
1	LeakyReLU(Linear)	$\dim(\mathbf{h})$	$\dim(\mathbf{h})$

Generator. We follow the original StyleGAN2 architecture. However, we use the latent vector $\mathbf{h}(t)$ instead of the intermediate latent code w of StyleGAN2.

Discriminator. The network architecture of the discriminator also uses that of StyleGAN2. However, our discriminator receives time (t) as a conditional input. To this end, we use the positional embedding of t as in Ho et al. (2020). The hyperparameters for the discriminator are in Table 9.

B.5 Training details

We train our model using the Adam optimizer for training both the generator and the discriminator. We use the exponential moving average (EMA) when training the generator, which achieves high performance in Ho et al. (2020); Song et al. (2021c); Karras et al. (2020a). The hyperparameters for the optimizer are in Table 9. The adversarial training object of our model is as follows:

$$\begin{aligned}&\min_{\phi} \mathbb{E}_{\mathbf{i}(t)\sim \mathbf{q}_{\mathbf{i}(t)}}\big[-\log(D_\phi(\mathbf{i}(t),t)) + \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(\mathbf{0},\sigma^2\mathbf{I})}\big[-\log(1-D_\phi(G_\theta(M_\psi(\mathbf{z})),t))\big]\big],\\ &\max_{\theta,\psi} \mathbb{E}_{\mathbf{i}(t)\sim \mathbf{q}_{\mathbf{i}(t)}}\big[\mathbb{E}_{\mathbf{z}\sim\mathcal{N}(\mathbf{0},\sigma^2\mathbf{I})}\big[\log(D_\phi(G_\theta(M_\psi(\mathbf{z})),t))\big]\big],\end{aligned} \tag{11}$$

where, $\mathbf{q}_{\mathbf{i}(t)}$ is the interpolated image distribution, D_ϕ is denoted as the discriminator, G_θ is denoted as the generator, and M_ψ is denoted as the mapping network of our model. We also use the R_1 regularization and the path length regularization (Karras et al., 2020b). λ_{R_1} (resp. λ_{path}) means the coefficient of the R_1 regularization term (resp. the coefficient of the path length regularization term). Each regularizer term is as follows:

$$R_1(\phi) = \lambda_{R_1}\mathbb{E}_{q(\mathbf{i}(t))}\big[\|\nabla_{\mathbf{i}(t)}(D_\phi(\mathbf{i}(t)|t))\ \|^2\big], \tag{12}$$

$$\text{Path length} = \lambda_{path}\mathbb{E}_{\mathbf{h}(t),\mathbf{i}(t)}\big(\|\mathbf{J}^T_{\mathbf{h}(t)}\mathbf{i}(t)\ \|_2 - a\big), \tag{13}$$

where $\mathbf{J}_{\mathbf{h}(t)} = \partial G_\theta(\mathbf{h}(t))/\partial\mathbf{h}(t)$ is the Jacobian matrix. The constant a is set dynamically during optimization to find an appropriate global scale. The path length regularization helps with the mapping from latent vectors to images. The lazy regularization makes training stable by computing the regularization terms (R_1, path length) less frequently than the main loss function. In SPI-GAN, the regularization term for the generator and the discriminator is calculated once every 4 iterations and once every 16 iterations, respectively. The hyperparameters for the regularizers are in Table 9.

B.6 Hyperparameters

We list all the key hyperparameters in our experiments for each dataset. Our supplementary material accompanies some trained checkpoints and one can easily reproduce.

Table 9: Hyperparameters set for SPI-GAN.

		CIFAR-10	CelebA-HQ-256	LSUN-Church-256
Augmentation	ADA target	0.6	0.6	0.6
	ADA interval	4	4	4
	λ_{mixing} (%)	0	90	90
Architecture	Mapping network	1	7	7
	Discriminator	Original	Residual	Residual
Optimizer	Learning rate for generator	0.0025	0.0025	0.0025
	Learning rate for discriminator	0.0025	0.0025	0.0025
	EMA	0.9999	0.999	0.999
	ODE Solver	4th order Runge–Kutta		
Regularization	Lazy generator	4	4	4
	Lazy discriminator	16	16	16
	λ_{R_1}	0.01	10	10
	λ_{path}	0	2	2

B.7 Sampling method of SPI-GAN

The sampling algorithm is similar to the training algorithm. The two main differences are i) we sample a set of noisy vectors $\{\mathbf{z}^l\}_{l=1}^N$ whereas we use $\{\mathbf{x}_T^l\}_{l=1}^N$ in the training algorithm, and ii) for sampling, we need only $\{\mathbf{h}^l(1)\}_{l=1}^N$.

Algorithm 2: How to sampling SPI-GAN

```
  Input: Noisy vectors $\mathbf{z}^l \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$
1 Sample a set of noisy vectors $\{\mathbf{z}^l\}_{l=1}^N$;
2 Calculate $\{\mathbf{h}^l(1)\}_{l=1}^N$ via the mapping network which processes $\{\mathbf{z}^l\}_{l=1}^N$ by solving Equation 6 after
    setting $j = K$;
3 Generate a fake image $\{\hat{\mathbf{i}}^l(1)\}_{l=1}^N$ with the generator;
4 return fake image $\{\hat{\mathbf{i}}^l(1)\}_{l=1}^N$;
```

C Visualization

We introduce several high-resolution generated samples.

C.1 CIFAR-10

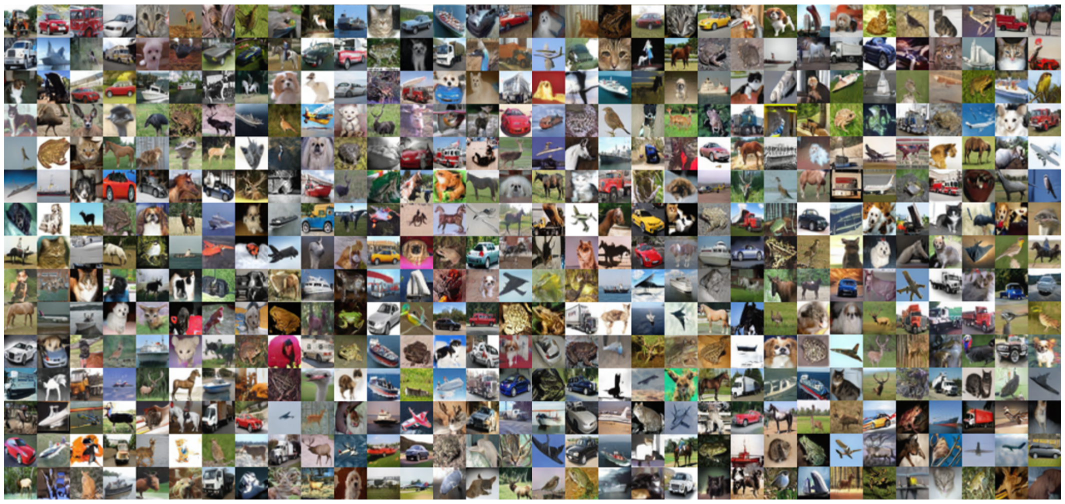

Figure 12: Qualitative results on CIFAR-10.

C.2 CelebA-HQ-256

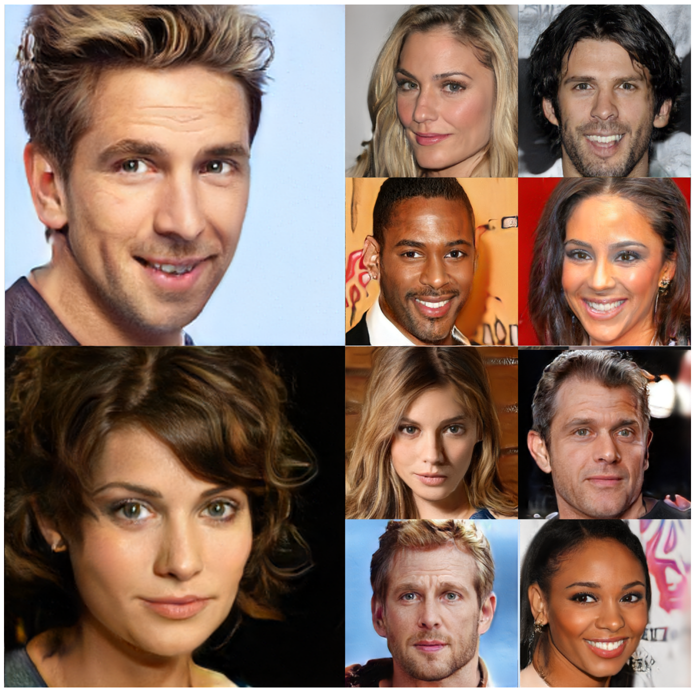

Figure 13: Qualitative results on CelebA-HQ-256.

C.3 LSUN-Church-256

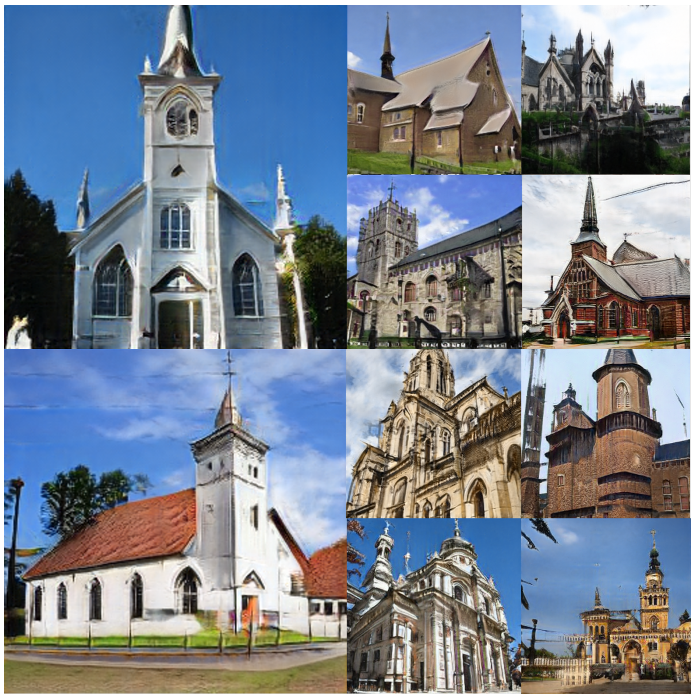

Figure 14: Qualitative results on LSUN-Church-256.